Covert Channel Phase 4 Report – Hamdi Alperen Karapınar

Covert Channel Type: Source and Destination Address Spoofing

1. Objective

The goal of Phase 4 was to implement a real-time mitigation mechanism in the middlebox processor that blocks covert UDP communications based on heuristic detection. This extends Phase 3's detector by actively preventing the exfiltration of covert messages and logging the outcome for forensic analysis.

Mitigation is triggered by source IP heuristics and is integrated seamlessly into the existing Docker + NATS-based architecture.

2. Architecture & Environment

Phase 4 was built on top of the Phase 3 setup, preserving:

- NATS for inter-container message passing
- python-processor container as the middlebox decision-making unit
- sec container for both normal and covert sender traffic
- insec container for receiving traffic

The main.py file was extended to drop packets identified as covert and log all dropped packets.

3. Mitigation Implementation

The following updates were made to main.py:

- Heuristic detection logic based on last octet of source IP (≠21 = covert)
- If a packet is detected as covert, it is:
  - Dropped (not forwarded to the receiver container)
  - Logged into a file mitigator.csv
  - Printed to terminal with [Mitigator] DROP prefix

- All packets (both dropped and forwarded) are logged with metadata (src_ip, length, inter_arrival, entropy, label). Later, dropped flag is added to them.

Example mitigation logic added:

#When mitigator is enabled by ENABLE_MITIGATION = TRUE

```
if ENABLE_MITIGATION and heuristic_pred:
    print(f"[Mitigator] Dropped suspected covert packet from {ip_src}")
    return
```

4. Engineering Challenges

4.1 Logging All Traffic Consistently

Initially, only covert (dropped) packets were being logged. This made it difficult to compare mitigated vs. passed traffic.

Solution: Logging was unified so both covert and normal packets are written to mitigator.csv, including classification outcome. Dropped flag helps to distinguish logged outputs.

code > python-processor > mitigator.csv > data

```
src_ip,length,inter_arrival,entropy,label,dropped
10.1.0.67,6,0,2.584962500721156,covert,True
10.1.0.67,6,0.0069239139556884766,2.584962500721156,covert,True
10.1.0.111,6,0.10229706764221191,2.584962500721156,covert,True
10.1.0.111,6,0.007579326629638672,2.584962500721156,covert,True
10.1.0.118,6,0.13695955276489258,2.584962500721156,covert,True
10.1.0.118,6,0.010089635848999023,2.584962500721156,covert,True
10.1.0.101,6,0.09818816184997559,2.584962500721156,covert,True
10.1.0.101,6,0.010418891906738281,2.584962500721156,covert,True
10.1.0.114,6,0.12934136390686035,2.584962500721156,covert,True
10.1.0.114,6,0.010678529739379883,2.584962500721156,covert,True
10.1.0.116,6,0.13649225234985352,2.584962500721156,covert,True
```

#Covert channel sender is executed first, that's why we only see the messages with the flag dropped is True. Implemented mitigator drops these packets.

4.2 File Permission Issues in Docker

Permission errors occurred when writing to mitigator.csv due to Docker file volume mappings.

Solution: All file paths were hardcoded to /code/python-processor/, which is the working directory inside the container. It was ensured that the required folder existed and was writable.

4.3 Ensuring Real-Time Behavior

The original delay-based simulation sometimes caused multiple packets to accumulate before detection.

Solution: Sleep intervals were recalibrated, and detection, logging, and drop decisions were made synchronously per packet.

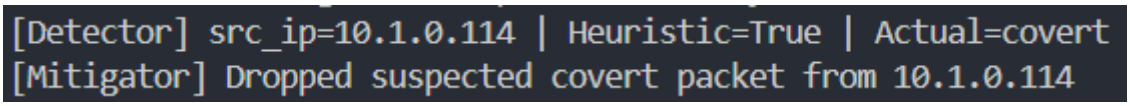
5. Experimental Demonstration

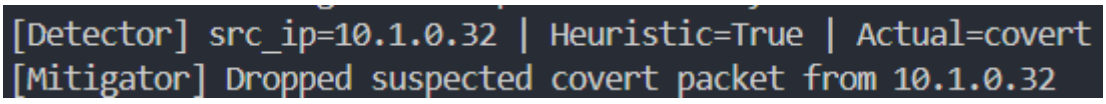
Using the same normal_sender.py and covert_sender_benchmark.py, the following steps were taken:

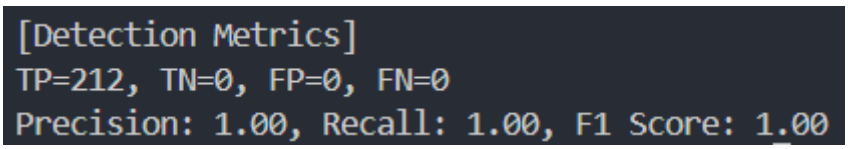
1. Both senders were run concurrently

2. Logs from main.py were monitored to observe packet drops in real time

3. It was verified that covert messages never reached the receiver







The mitigator log mitigator.csv contains complete trace data.

6. Benchmark Evaluation

In post-mitigation evaluation, original messages were compared with those received by receiver.py. It was confirmed that:

- All covert messages were blocked

- No message leakage occurred

Additional experimentation used the experiment_runner.py and evaluate_confidence.py pipelines with mitigation enabled. The dataset showed 100% recall, with covert messages missing entirely from the receiver side.

```
10.1.0.115,6,0.009507417678833008,2.584962500721156,covert,True
10.1.0.115,6,0.12504363059997559,2.584962500721156,covert,True
10.1.0.115,6,0.009794950485229492,2.584962500721156,covert,True
10.1.0.101,6,0.1788187026977539,2.584962500721156,covert,True
10.1.0.101,6,0.008303403854370117,2.584962500721156,covert,True
10.1.0.115,6,0.15848374366760254,2.584962500721156,covert,True
10.1.0.115,6,0.009145975112915039,2.584962500721156,covert,True
10.1.0.46,6,0.1207726001739502,2.584962500721156,covert,True
10.1.0.46,6,0.007036685943603516,2.584962500721156,covert,True
10.1.0.21,21,1.6985962390899658,4.392317422778761,normal,False
10.1.0.21,21,0.005164146423339844,4.392317422778761,normal,False
10.1.0.21,97,0.0965118408203125,6.138522249381319,normal,False
10.1.0.21,97,0.01080465316772461,6.138522249381319,normal,False
10.1.0.21,26,0.08825016021728516,4.546593564294938,normal,False
10.1.0.21,26,0.006639719009399414,4.546593564294938,normal,False
10.1.0.21,96,0.09640312194824219,6.397462500721156,normal,False
10.1.0.21,96,0.007575273513793945,6.397462500721156,normal,False
10.1.0.21,44,0.0898275375366211,5.3230679822736615,normal,False
10.1.0.21,44,0.0067408084869384766,5.3230679822736615,normal,False
```

Screenshot of the mitigator.csv log file tos how that normal packets are not dropped, unlike the packets transmitted in the covert channel.

7. File Summary

- main.py: Real-time detection + mitigation logic + logging

- mitigator.csv: Logs of all packets (dropped or passed) with features and labels

- covert_sender_benchmark.py: Sends obfuscated covert messages

- normal_sender.py: Sends regular UDP messages

- experiment_runner.py: Used to test full pipeline with mitigation

8. Conclusion

Phase 4 completed the real-time prevention pipeline for covert channel exfiltration. The middlebox processor can now:

- Detect covert messages using protocol-aware heuristics

- Drop and log packets based on detection results

- Record all traffic in a structured CSV for analysis.

Mitigation was validated under controlled traffic with high confidence. Combined with the detector from Phase 3, the system now functions as a full covert channel firewall capable of not just observing but also intervening in suspicious UDP traffic.

9. GitHub Repository

https://github.com/alperenkrpnr/middlebox