Phase 2 — Covert Channel via UDP Length Field Manipulation

1. Introduction

In this phase, we designed and implemented a covert channel mechanism based on **UDP length field manipulation**. The goal was to encode and transmit secret messages from a sender inside the sec container to a receiver in the insec container, by varying the payload length of UDP packets. The covert channel exclusively relies on modifying the packet length without altering other header fields, in compliance with the selected channel type: *Length Field Manipulation: Adjusting the UDP length field to encode information*.

Our design builds on the Phase 1 development environment, using Docker containers and the NATS-based middlebox (mitm) architecture. The covert sender (sender.py) and receiver (receiver.py) scripts are implemented in Python and communicate using standard UDP sockets. To ensure experimental reproducibility and performance evaluation, a benchmark_test.py script was developed to automate repeated tests under configurable parameters.

2. Architecture and Network Routing

While the sec, insec, and mitm containers were provided as part of the base environment, establishing communication between sec and insec required additional setup due to the use of distinct **macvlan networks** (routed and exnet). These networks isolate containers as if they are physical hosts, which prevents direct container-to-container communication.

To enable communication, the mitm container—connected to both networks—was configured as a router. The following steps were applied inside the mitm container:

- IP forwarding was enabled:
  ```
  sysctl -w net.ipv4.ip_forward=1
  ```
- Bidirectional forwarding was allowed via iptables:
  ```
  iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
  iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
  ```
- Static routing was added to the sec and insec containers:
  ```
  #from sec
  ip route add 10.0.0.0/16 via 10.1.0.2
  #from insec
  ip route add 10.1.0.0/16 via 10.0.0.2
  ```

This configuration allowed UDP packets sent by sender.py from sec to be routed via mitm and received by receiver.py inside insec.

3.Methodology

Message Encoding (Sender Side)

Each character in the message is encoded into an 8-bit binary string. This bitstream is then divided into randomly sized chunks of **1, 2, or 3 bits**. Each chunk corresponds to a unique UDP payload length, defined by a base length parameter (base_len) and an offset.

For example:

- `base_len + 0–1 → 1-bit values (0, 1)`
- `base_len + 2–5 → 2-bit values (00, 01, 10, 11)`
- `base_len + 6–13 → 3-bit values (000 to 111)`

The actual UDP packet is filled with dummy data (b'A' * length) and sent to the receiver.

A header is also sent before the message, using 8 length-encoded bits to indicate the number of characters expected by the receiver.

To further obfuscate the transmission pattern and resist traffic analysis, the sender randomly selects whether to encode 1, 2, or 3 bits for each chunk of the bitstream. This adaptive bit-length encoding ensures that the length values of packets vary in a non-deterministic manner, making detection more difficult.

Message Decoding (Receiver Side)

The receiver listens on a fixed UDP port and first extracts the header to determine the number of characters in the message. Then it reads each incoming packet and decodes its length into the original bit chunk (1, 2, or 3 bits), appending bits until 8-bit sequences are complete and can be decoded into ASCII characters.

The decoded message is printed and appended to a file named received_messages.txt.

Benchmarking

To evaluate performance, we implemented a benchmarking script (benchmark_test.py) which:

- Repeatedly sends messages with various parameters (base_len, delay, message content)
- Measures total transmission time
- Verifies message correctness by reading received_messages.txt
- Calculates:
    - Average transmission time
    - Covert channel capacity in bits per second
    - 95% confidence intervals
    - Message success rate

4.Test Parameters

The covert channel experiments were conducted using a set of parameters designed to evaluate the system's performance across different message lengths, base payload sizes, and transmission speeds. The goal was to observe the impact of these factors on capacity, decoding accuracy, and reliability.

- **Base Lengths (base_lengths)**: [50, 60]
  These values represent the initial UDP payload lengths used as the reference point for encoding bits via length manipulation. By testing with two distinct base lengths, we aimed to assess how shifting the length range affects decoding consistency and channel throughput.
- **Delays (delays)**: [0.05, 0.1] (seconds)
  These delays represent the time interval between consecutive packets. A shorter delay allows for faster transmission but increases the risk of congestion or packet collisions. A longer delay improves reliability but reduces throughput. Both were tested to strike a balance between speed and robustness.
- **Messages (messages)**:
    - "Hello" # A short, 5-character message
    - "This is a longer test message. " # A longer sentence (33 characters) used to simulate more substantial covert transmissions

  These messages were selected to test both ends of the spectrum: minimal payloads and longer, sentence-style content. The longer message also helps evaluate how decoding behaves over a sustained sequence of bit chunks.
- **Repetitions (repeats)**: 10
  Each test scenario was repeated 10 times to allow statistical evaluation of the results. This enabled the computation of average transmission time, covert channel capacity (in bits per second), and 95% confidence intervals for each configuration.

5.Results

Here are representative results from the benchmark script:

All messages were correctly received in each case. The results confirm that the covert channel performs reliably under the tested parameters, and provides reasonable throughput given the added obfuscation through randomized bit group sizes.

Base Length	Delay (s)	Message	Avg Time (s)	Capacity (bps)	95% CI	Success Rate
50	0.05	Hello	1.51	26.50	±0.03	100%
50	0.05	This is a longer test message.	6.59	36.45	±0.11	100%
50	0.1	Hello	2.58	15.48	±0.12	100%
50	0.1	This is a longer test message.	12.47	19.25	±0.25	100%
60	0.05	Hello	1.50	26.61	±0.05	100%
60	0.05	This is a longer test message.	6.42	37.37	±0.18	100%
60	0.1	Hello	2.41	16.59	±0.08	100%
60	0.1	This is a longer test message.	12.63	19.00	±0.30	100%

6.Conclusion

Our implementation of a UDP length-based covert channel demonstrated the feasibility of transmitting hidden information by manipulating packet lengths alone. Through careful design and testing, we achieved reliable communication between isolated network environments with 100% message delivery success rate across all test configurations.

The results reveal several important characteristics of our covert channel implementation:

1. **Performance tradeoffs**: The delay parameter significantly impacted throughput, with 0.05-second delays achieving approximately double the capacity (26-37 bps) compared to 0.1-second delays (15-19 bps).
2. **Scalability**: Longer messages achieved higher bits-per-second rates than shorter messages under identical conditions, suggesting efficient overhead amortization during extended transmissions.
3. **Base length influence**: The base length parameter (50 vs. 60) had minimal impact on overall performance, indicating the robustness of our encoding scheme across different payload size ranges.
4. **Stealth characteristics**: The randomized bit grouping approach (1, 2, or 3 bits per packet) creates non-deterministic length patterns that enhance resistance to statistical traffic analysis while maintaining reliable decoding.

This implementation demonstrates that even with the constraints of using only packet length manipulation, effective covert channels can be established that balance throughput, reliability, and detection resistance. Future work could explore adaptive delay mechanisms, more sophisticated encoding schemes, or countermeasures against length-based covert channel detection systems.

7.GitHub Repository

GitHub repository: https://github.com/rab-ai/middlebox