Covert Channel Phase 2 Report – Hamdi Alperen Karapınar
Covert Channel Type: Source and Destination Address Spoofing

1. Introduction

In Phase 2, a covert communication channel was implemented using source IP address spoofing, encoding data by modifying the last octet of the source IP in outgoing UDP packets.

The covert sender and receiver were deployed in the sec and insec containers respectively. A benchmarking framework was implemented to measure average message transmission time, 95% confidence intervals, and channel capacity.

2. Implementation Overview

Sender (sender.py) Explanation:

- **Encoding strategy:** Each character in the message is encoded as the last octet of the source IP address.
- **Packet crafting:** Uses Scapy to spoof the source IP address (e.g., 10.1.0.X, where X = ord(char)).
- **Parametrization:** Accepts message content, delay, destination IP, and port as arguments.

```
spoofed_ip = f"10.1.0.{ord(char)}"

pkt = IP(src=spoofed_ip, dst=dst_ip)/UDP(sport=12345, dport=dst_port)/b"covert"

send(pkt)

# Note: `b"covert"` defines a raw byte string payload required by Scapy.
```

Receiver (receiver.py) Explanation:

- **UDP Listener:** Binds to a specified port and listens for packets.
- **Decoding:** Extracts the last octet of the sender's IP and converts it back to a character using chr().

Benchmark (benchmark.py) Explanation:

- Automatically starts the receiver.
- Repeatedly invokes the sender in a subprocess.

- Logs total transmission time for each trial.
- Calculates mean, 95% confidence interval, and channel capacity.
- Writes results to a CSV file.

3. Comparative Benchmark Results

To better understand the effects of message length and inter-packet delay on covert channel performance, three experiments were conducted:

A. Delay Comparison (Same Message):

Metric	**Delay = 0.1s**	**Delay = 0.3s**
Message	'Alperen's Covert Channel'	'Alperen's Covert Channel'
Characters (bits)	24 (192 bits)	24 (192 bits)
Avg. Total Time	5.853 sec	10.608 sec
95% Confidence Interval	(5.552, 6.154) sec	(10.212, 11.004) sec
Estimated Capacity	32.80 bits/sec	18.10 bits/sec

B. Message Length Comparison (Same Delay = 0.1s):

Metric	**Short Message: 'Alperen'**	**Long Message: 'Alperen's Covert Channel'**
Characters (bits)	7 (56 bits)	24 (192 bits)
Avg. Total Time	2.006 sec	5.853 sec
95% Confidence Interval	(1.787, 2.225) sec	(5.552, 6.154) sec
Estimated Capacity	27.92 bits/sec	32.80 bits/sec

C. Summary

- **Longer delays** lead to significantly **lower capacity**, though the impact becomes more pronounced with longer messages.
- **Longer messages** do not necessarily reduce capacity proportionally. Since certain fixed costs like setup time or receiver initialization remain constant regardless of message size, longer messages can result in a more efficient use of transmission time overall.

4. Why IP Spoofing?

The chosen method is effective for covert communication because it embeds data in a header field not typically used for integrity checks by many firewalls and monitoring systems. Unlike payload-based channels, which are easier to inspect and filter, header manipulation (like IP spoofing) allows transmission of data in a stealthier and less detectable way.

In my implementation, the Python ord() function is used to convert each character in the secret message into its ASCII integer value (e.g., ord('A') = 65). This value is then used as the final octet of the spoofed IP address, resulting in IPs like 10.1.0.65 for the character 'A'. This approach allows precise, byte-level control over the encoded content.

On the receiver side, the last octet of each incoming packet's source IP is extracted and passed through the chr() function to decode it back into a readable character. This simple mapping ensures an accurate and lossless reconstruction of the original message.

I specifically chose to spoof only the last octet of the IP address because:

- It ensures that the spoofed IP remains within the Docker macvlan subnet (e.g., 10.1.0.X), avoiding routing issues.
- Docker-based networks may block or reject traffic from invalid or non-local IP ranges.
- This design maintains simplicity while achieving an effective 8 bits per packet throughput.

While it is possible to encode more bits using other octets or combining with ports, focusing on the last octet ensured reliable transmission and straightforward decoding without complex synchronization or reassembly logic.

5. System Configuration & Fixes

Problem: Initial packets were not routed between sec and insec containers.

Solution:

In mitm container:

```
sysctl -w net.ipv4.ip_forward=1

iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT

iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
```

Explanation: Enables packet forwarding between interfaces and allows the MITM container to route traffic between secure and insecure networks.

6. Scapy Installation Without Internet

Because the container lacked internet access, Scapy was installed manually:

1. Scapy was downloaded locally and copied:

```
docker cp ../scapy-2.6.1/ sec:/code/sec/
```

2. Then installed inside the container:

```
cd /code/sec/scapy-2.6.1

python3 setup.py install
```

7. Design Justification

- **Source IP spoofing:** Offers a simple and low-bandwidth covert channel using deterministic IP manipulation.
- **Delay parameter:** Used to introduce inter-packet delays and reduce detectability.
- **Benchmarking:** Enables consistent benchmarking of performance metrics (mean time, confidence interval, and capacity).
- **Error Handling:** Negative durations are filtered; sender subprocess failures are skipped.

8. GitHub Repository

https://github.com/alperenkrpnr/middlebox