TPPhase2: TCP Urgent Pointer Covert Channel Implementation and Analysis

Batuhan Teberoğlu — 2581056

April 15, 2025

1 Introduction

This report documents the implementation and performance analysis of a covert channel using the TCP Urgent Pointer field. The goal was to create a hidden communication channel by embedding data in the TCP Urgent Pointer field.

2 TCP Urgent Pointer Field Overview

2.1 What is the TCP Urgent Pointer Field?

It is possible to notify the receiver for an upcoming urgent data as part of the TCP payload. Sender expects the receiver to take action immediately, skipping the usual processing queue. An example to a use case is when the user aborts a process, the change in status could be sent as an urgent data.[1]

TCP segment contains two fields related to urgent data: 1 bit URG flag and 16 bit Urgent Pointer. When the URG flag is set, the Urgent Pointer field indicates the sequence number of the byte following the urgent data. An interesting property of this field is that it can contain values even when the URG flag is not set. This characteristic creates an opportunity for covert data transmission.

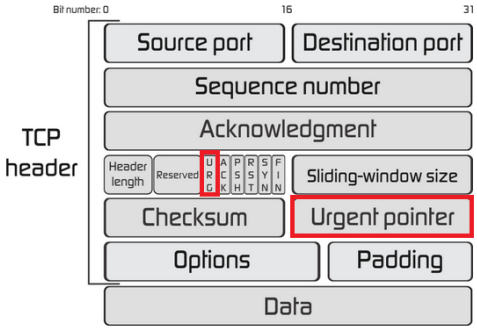

Figure 1: TCP Transmission Control Protocol Segments and Fields. Source: FiberBit

2.2 Advantages of Using the Urgent Pointer for Covert Channels

[2]

- **Available space**: The field provides 16 bits (2 bytes) of space per packet for covert data, much less if it is restricted to be near sequence numbers.
- **Difficulty of implementing**: It is easy to implement, just replace the Urgent Pointer field bits.

- **Availability**: It is advised to not implement the Urgent mechanism in applications, [4] Urgent Pointer will mostly be available for use for covert channel.

2.3 Limitations and Challenges

[2]

- **Network interference**: Middleboxes like firewalls or NAT devices might normalize TCP headers, potentially corrupting the covert data.
- **Detection**: As Urgent mechanism is rarely used, consistent use of it within covert channel may raise alerts.

3 Implementation

3.1 System Components

My implementation consists of two Python scripts:

- **tppphase2_sender.py**: Encodes data into the Urgent Pointer field of outgoing TCP packets
- **tppphase2_receiver.py**: Captures TCP packets and extracts the covert data

3.2 Sender Implementation

The sender program reads input data, optionally encrypts it and sends it by embedding chunks in the Urgent Pointer field of outgoing TCP packets. Key features include:

- Configurable bit width (1-16 bits) for the covert data
- Optional Fernet symmetric encryption with a pre-shared key
- Configurable timing between packets

3.3 Receiver Implementation

The receiver captures incoming TCP packets, extracts the covert data from the Urgent Pointer field, and reassembles the original message. Key features include:

- Bit-level accumulation to reassemble the original data
- Optional decryption of the received data

3.4 Encryption

If the covert channel is discovered, intercepting the communication becomes straightforward. To mitigate this risk, the sent data can be encrypted.
In this implementation, both the sender and receiver use symmetric encryption with a pre-shared key. It uses the Fernet encryption method from Python's cryptography library.

4 How to Run the Experiment

4.1 Processor Setup

Exec into python-processor container and run:

```
python3 tppphase2_main.py
```

4.2 Receiver (insec) Setup

Exec into insec container and run:

```
python3 tppphase2_receiver.py --output ./received_files --bits 16 --decrypt
```

Available parameters:

- `--output`: Directory to save received files (default: ./received_files)
- `--bits`: Number of bits to use from the urgent pointer (1-16, default: 16)
- `--decrypt`: Enable decryption of received data (optional)

4.3 Sender (sec container) Setup

Exec into sec container and run:

```
python3 tppphase2_sender.py --file ./secret_message.txt --interval 0.2 --bits 16 --encrypt
```

Available parameters:

- `--file`: Path to the file to send (required)
- `--interval`: Delay between packets in seconds (default: 0.2)
- `--bits`: Number of bits to use in the urgent pointer (1-16, default: 16)
- `--encrypt`: Enable encryption of data (optional)

4.4 Experiment Scenarios

For a comprehensive performance analysis, I ran the following experiment scenarios:

- Different bit widths: 4, 8, 12, and 16 bits
- Various packet intervals: 0.1, 0.2, 0.5, and 1.0 seconds
- File size fixed 1KB
- With and without encryption

5 Results

5.1 Channel Capacity

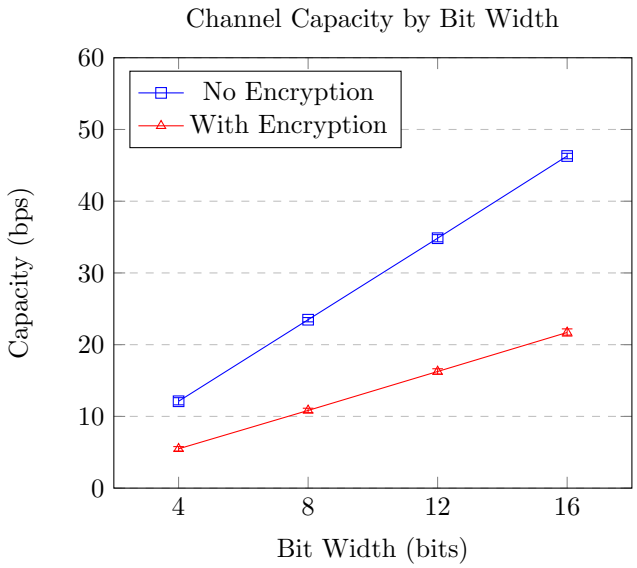

Figure 2: Channel capacity as a function of bit width (conf. interval $<$ +-0.5 shown in graph)

Table 1: Measured Channel Capacity at Different Bit Widths

Bit Width (bits)	**No Encryption (bps)**	**With Encryption (bps)**
4	12.11 ± 0.5	5.48 ± 0.31
8	23.48 ± 0.3	10.83 ± 0.31
12	34.84 ± 0.5	16.26 ± 0.38
16	46.30 ± 0.38	21.70 ± 0.5

5.2 Impact of Packet Interval

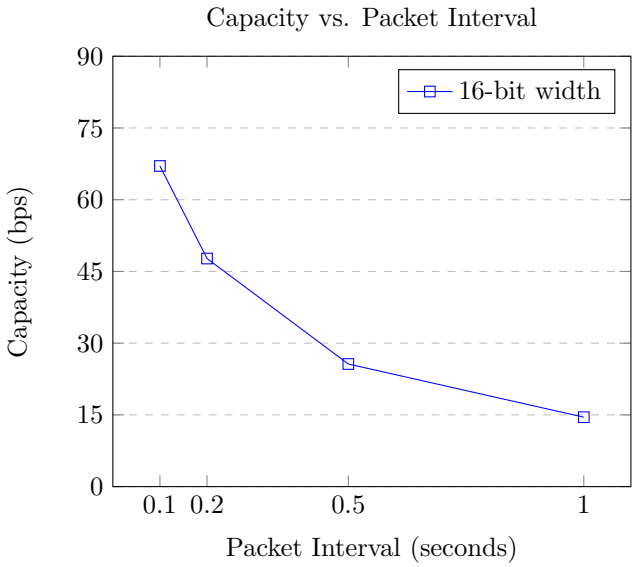

Figure 3: Effect of packet interval on channel capacity

Table 2: Measured Capacity at Different Packet Intervals (16-bit width)

Packet Interval (s)	**Capacity (bps)**
0.1	67.05
0.2	47.70
0.5	25.64
1.0	14.52

5.3 Comments

- Using more bits in the Urgent Pointer improves channel capacity.
- Encryption offers security but reduces effective throughput.
- Smaller packet intervals maximize data transfer rate but may increase detectability.

6 Potential Improvements

Based on my experiments and research, I have identified several ways to improve this covert channel implementation:

6.1 Enhancing Stealth

- **Traffic pattern matching**: Adjust packet timing to match patterns of common applications like web browsing or streaming.
- **Random intervals**: Add jitter to packet timing to make traffic appear more natural.
- **Multiple destination ports**: Use different ports for transmission to avoid pattern detection.

6.2 Performance Enhancements

- **Multiple connections**: Use several TCP connections in parallel to increase throughput.
- **Data compression**: Compress data before transmission to reduce volume.

6.3 Security Improvements

- **Dynamic key exchange**: Replace the hardcoded key with a secure key exchange mechanism.
- **Time-based rotation**: Periodically change encryption keys or encoding methods.

References

[1] NetworkLessons, *TCP Urgent Pointer Field*, Available at: https://notes.networklessons.com/tcp-urgent-pointer-field

[2] Hintz, *Covert Channels in TCP/IP*, DEFCON 10 Presentation, Available at: http://www.defcon.org/images/defcon-10/dc-10-presentations/dc10-hintz-covert.ppt

[3] "Transmission Control Protocol," RFC 793, 1981.

[4] "On the Implementation of the TCP Urgent Mechanism," RFC 6093, 2011.

[5] S. Zander, G. Armitage, P. Branch, "A survey of covert channels and countermeasures in computer network protocols," IEEE Communications Surveys & Tutorials, vol. 9, no. 3, pp. 44-57, 2007.