TCP Urgent Pointer Covert Channel Implementation, Detection, and Mitigation

Batuhan Teberoğlu — 2581056

June 15, 2025

1 Introduction

This report documents the complete lifecycle of a TCP Urgent Pointer covert channel project, including implementation (Phase 2), detection (Phase 3), and mitigation (Phase 4). The goal was to create a hidden communication channel by embedding data in the TCP Urgent Pointer field, then develop effective countermeasures against such channels.

2 Phase 2: Covert Channel Implementation

2.1 TCP Urgent Pointer Field Overview

2.1.1 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 by setting the Urgent Pointer field to covert data values while keeping the URG flag unset, making the channel more stealthy.

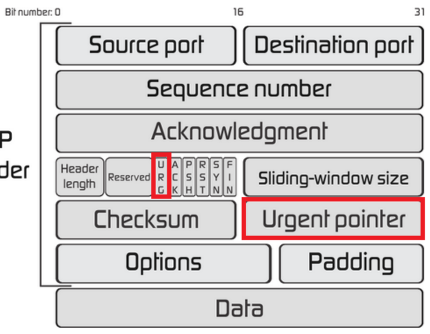

Figure 1: TCP Transmission Control Protocol Segments and Fields. Source: FiberBit

2.1.2 Advantages of Using the Urgent Pointer for Covert Channels

- **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. [2]
- **Availability**: It is advised to not implement the Urgent mechanism in applications, [4] Urgent Pointer will mostly be available for use for covert channel.

2.1.3 Limitations and Challenges

- **Network interference**: Middleboxes like firewalls or NAT devices might normalize TCP headers, potentially corrupting the covert data. [2]
- **Detection**: While the URG flag is rarely used, setting the Urgent Pointer field without the URG flag makes detection more difficult as the field appears unused. [2]

2.2 Implementation

2.2.1 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

2.2.2 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 without setting the URG flag, following Hintz's stealth technique. 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

2.2.3 Receiver Implementation

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

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

2.2.4 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.

2.2.5 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
- With and without encryption

2.2.6 Test File Specification

For all experiments, we used a standardized test file (`secret_message.txt`) containing exactly 100 characters of Lorem ipsum text. This file size was specifically chosen to enable rapid test execution while providing sufficient data to measure covert channel performance accurately.

The 100-character limit serves several important purposes:

- **Fast test execution**: Enables quick iteration during parameter testing and validation
- **Consistent baseline**: Provides uniform data size across all bit-width and interval variations
- **Manageable transmission time**: Prevents excessively long test runs that could introduce network variability
- **Sufficient data volume**: Large enough to measure throughput and capacity metrics meaningfully

It should be noted that using larger files may significantly impact the experimental results. Larger files would:

- Increase transmission duration, potentially introducing more network jitter and timing variations
- Amplify the effects of packet loss or reordering on covert channel reliability
- Change the statistical characteristics that detection algorithms rely upon
- Affect the measured capacity and throughput metrics due to protocol overhead becoming proportionally smaller

3 How to Run Experiments

This section describes how to execute experiments for all phases using the integrated testing framework.

3.1 Prerequisites

Ensure the Docker environment is running with all containers:

```
docker compose up -d
```

Verify that all required containers are running:

- `python-processor`: Main processing container
- `sec`: Secure network container (sender)
- `insec`: Insecure network container (receiver)
- `nats`: Message broker for packet forwarding

3.2 Using the Testing Framework

The project includes a comprehensive testing framework (`run_test.py`) that automates experiments for all phases:

```
python3 run_test.py
```

3.2.1 Interactive Test Selection

The framework provides an interactive menu for processor selection:

1. **TPPhase2 - Covert Channel**: Tests the basic covert channel implementation
2. **TPPhase3 - Covert Channel Detector**: Tests detection capabilities
3. **TPPhase4 - Covert Channel Mitigator**: Tests mitigation effectiveness

3.2.2 Test Execution Options

For each processor type, the framework offers three execution modes:

- **Run all scenarios**: Executes all parameter combinations automatically
- **Run single test**: Allows selection of specific parameter combinations for debugging
- **Test docker commands only**: Validates Docker container connectivity and basic functionality

3.2.3 Test Scenarios

The framework automatically runs experiments with the following parameters:

- **Bit widths**: 4, 8, 12, and 16 bits
- **Packet intervals**: 0.1, 0.2, 0.5, and 1.0 seconds
- **Encryption**: Both encrypted and unencrypted variants
- **Traffic environment**: Background legitimate traffic (Phases 3 & 4) or isolated covert channel (Phase 2)

3.2.4 Results and Output

Test results are automatically saved to the `test_results/` directory:

- **JSON reports**: Detailed metrics and performance data
- **Processor-specific results**: Detection metrics (Phase 3) or mitigation statistics (Phase 4)
- **Summary statistics**: Success rates, transmission times, and capacity measurements

3.2.5 Manual Execution

For manual testing or debugging, individual components can be run separately:

Phase 2 Manual Setup:

```
# Processor (python-processor container)
python3 tppphase2_main.py

# Receiver (insec container)
python3 tppphase2_receiver.py --output ./received_files --bits 16 --decrypt

# Sender (sec container)
python3 tppphase2_sender.py --file ./secret_message.txt --interval 0.2 --bits 16 --encrypt
```

Phase 3 Manual Setup:

```
# Detector (python-processor container)
python3 tppphase3_main.py

# Then run Phase 2 sender/receiver as above
```

Phase 4 Manual Setup:

```
# Mitigator (python-processor container)
python3 tppphase4_main.py

# Then run Phase 2 sender/receiver as above
```

3.3 Phase 2 Results

3.3.1 Channel Capacity

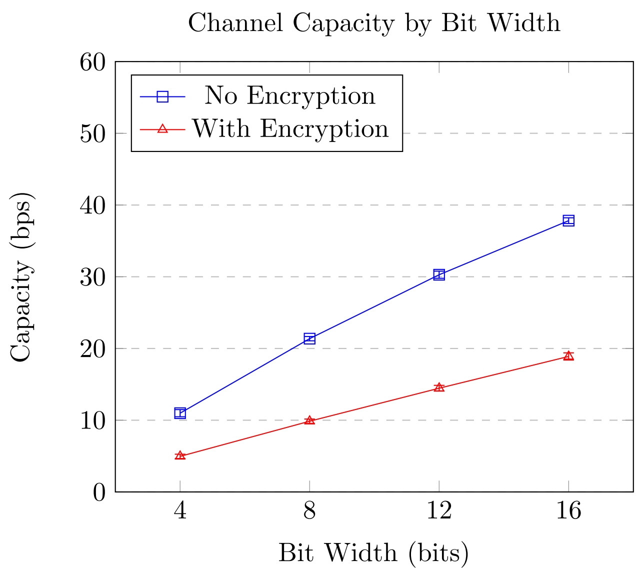

Figure 2: Channel capacity as a function of bit width (conf. interval < +-0.5 shown in graph)

Table 1: Measured Channel Capacity at Different Bit Widths

Bit Width (bits)	No Encryption (bps)	With Encryption (bps)
4	10.98 ± 0.5	4.96 ± 0.3
8	21.37 ± 0.3	9.86 ± 0.3
12	30.28 ± 0.5	14.46 ± 0.4
16	37.82 ± 0.4	18.89 ± 0.5

3.3.2 Impact of Packet Interval

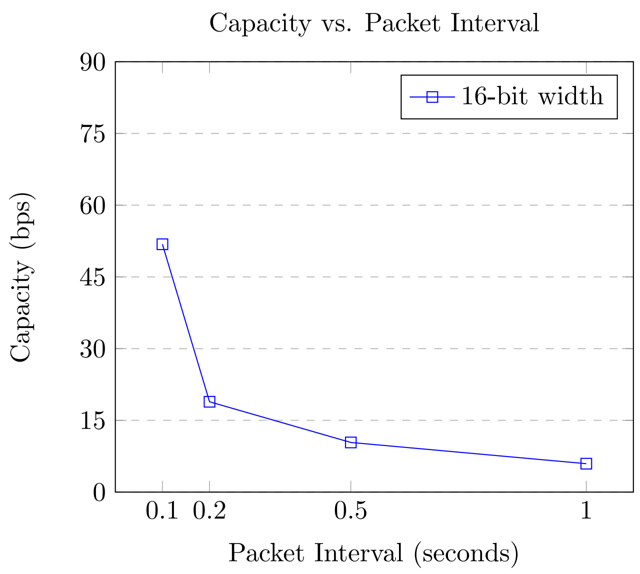

Figure 3: Effect of packet interval on channel capacity

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

Packet Interval (s)	**Capacity (bps)**
0.1	51.83
0.2	18.89
0.5	10.38
1.0	5.94

3.3.3 Phase 2 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.

4 Phase 3: Covert Channel Detection

4.1 Detection Approach

4.1.1 Detection System Architecture

The detection system (`tppphase3_main.py`) consists of:

- **NATSDetector**: Main detector class that monitors NATS packet streams
- **CovertChannelDetector**: Detection engine with configurable thresholds
- **Real-time classification**: Each packet is classified as either legitimate or covert

4.1.2 Detection Rules

The detector implements URG-based detection rules focusing on protocol violations:

- **Rule 1**: URG flag with suspicious pointer patterns - URG pointer in ASCII range (32-126) suggests data encoding
- **Rule 2**: URG flag with minimal payload - URG pointer with payload length 1 byte indicates potential covert data
- **Rule 3**: URG pointer without URG flag - Clear protocol violation where urgent pointer is set without the corresponding flag
- **Rule 4**: Statistical entropy analysis - High entropy in URG pointer values over time indicates data encoding

4.2 Implementation

4.2.1 Monitoring Process

The detector operates through the following workflow:

1. Subscribes to NATS topics (`inpktsec`, `inpktinsec`) for packet monitoring
2. Analyzes each packet structure to extract TCP header information
3. Classifies packets as legitimate or covert based on URG field analysis
4. Performs anomaly detection using the CovertChannelDetector engine
5. Forwards all packets transparently with proper TCP checksum recalculation
6. Generates real-time detection alerts and maintains session statistics

4.2.2 Packet Classification Logic

The classification system focuses specifically on URG field violations rather than port-based heuristics:

- Normal TCP traffic without URG field manipulation → **LEGIT**
- Packets with URG pointer in ASCII range (32-126) → **COVERT**
- Packets with URG flag but minimal payload → **COVERT**
- Packets with URG pointer but no URG flag → **COVERT**

4.3 Detection Experiment

4.3.1 Experimental Setup

A detection experiment was conducted with the following configuration:

- **Detection threshold**: 0.5
- **Entropy threshold**: 0.3
- **Monitoring duration**: 24.7 seconds
- **Traffic mix**: Combination of legitimate TCP traffic and TPPhase2 covert channel

4.3.2 Traffic Composition

The experimental session captured:

- **Total packets processed**: 354
- **URG flag packets**: 50 (14.1%)
- **Packets per second**: 14.3

4.4 Phase 3 Results

4.4.1 Perfect Detection Performance

The detector achieved exceptional performance with perfect classification:

Table 3: Detection Performance Results

Metric	Value
Accuracy	1.000 (100%)
Precision	1.000 (100%)
Recall	1.000 (100%)
F1-Score	1.000 (100%)
False Positive Rate	0.000 (0%)

4.4.2 Confusion Matrix Analysis

The detector achieved perfect classification with zero errors:

Table 4: Detection Confusion Matrix

Classification	Count
True Positives (TP)	51
True Negatives (TN)	265
False Positives (FP)	0
False Negatives (FN)	0
Total Analyzed Packets	316

4.4.3 Statistical Analysis

The URG pointer analysis revealed clear patterns distinguishing covert from legitimate traffic:

Table 5: URG Pointer Statistical Analysis

Metric	Value
URG Mean Value	7068.06
URG Standard Deviation	13320.64
URG Minimum Value	0
URG Maximum Value	57005
URG Entropy	0.447
Anomaly Rate	16.1%

4.4.4 Packet Classification Results

The classifier successfully distinguished between legitimate and covert traffic:

Table 6: Packet Classification Summary

Classification	Count	Percentage
Covert Packets	51	16.1%
Legitimate Packets	265	83.9%
Total Packets	316	100%

4.5 Phase 3 Comments

- The detector achieved perfect performance
- URG-based detection rules proved highly effective, successfully identifying all covert channel packets without false positives

5 Phase 4: Comprehensive TCP URG Pointer Covert Channel Mitigation

5.1 Mitigation Approach

Building upon the successful detection capabilities demonstrated in Phase 3, Phase 4 implements a comprehensive covert channel mitigation system that not only detects but actively mitigates TCP URG pointer covert channels.

5.1.1 Mitigation Architecture

The mitigation system consists of several integrated components:

- **Detection Engine**: Detection using statistical analysis, entropy calculation, and Markov chain analysis
- **Configurable Mitigation Strategies**: Three distinct mitigation modes with different levels of intervention
- **Performance Benchmarking**: Comprehensive metrics collection for latency impact and throughput analysis
- **Statistical Analysis**: Shannon entropy calculation and covert channel capacity measurement

5.1.2 Detection and Mitigation Strategies

The system implements three configurable mitigation modes:

- **Mode 1 - Normalize**: Clear urgent pointer field when URG flag is not set (protocol compliance)
- **Mode 2 - Clear**: Clear both URG flag and pointer completely for maximum security
- **Mode 3 - Randomize**: Randomize urgent pointer values for suspicious packets to corrupt covert data

5.1.3 Advanced Detection Rules

Enhanced detection capabilities include:

- **Protocol Violation Detection**: URG pointer set without URG flag (confidence: 0.8)
- **ASCII Range Detection**: URG pointer values in printable range 32-126 (confidence: 0.6)
- **Minimal Payload Detection**: URG flag with payload 1 byte (confidence: 0.5)
- **Rate-based Detection**: High frequency URG flag usage (configurable threshold)
- **Statistical Anomaly Detection**: Shannon entropy analysis of URG pointer values
- **Markov Chain Analysis**: Unusual state transition patterns in URG pointer sequences

5.2 Implementation

5.2.1 Testing Framework

A comprehensive testing framework was developed to evaluate mitigation effectiveness across multiple scenarios:

- **11 Test scenarios**: Different bit widths (4, 8, 12, 16 bits) and packet intervals (0.1s, 0.2s, 0.5s, 1.0s)
- **Mixed traffic environment**: Background legitimate traffic combined with covert channel traffic
- **Encryption variations**: Tests with and without Fernet encryption
- **Real-time processing**: Live packet processing with immediate mitigation actions

5.2.2 Performance Metrics Collection

The system collects comprehensive performance data:

- Total packets processed and suspicious packets detected
- Detection rate and mitigation action counts
- Processing latency (mean and 95th percentile)
- Covert channel capacity reduction measurements
- Statistical entropy analysis and anomaly type distribution

5.3 Phase 4 Results

5.3.1 Mitigation Effectiveness

The mitigation system was tested against live TPPhase2 covert channel traffic with the following results:

Table 7: Phase 4 Mitigation Performance Metrics

Metric	Value
Total Packets Processed	119
Suspicious Packets	87
Mitigation Actions	50
Detection Rate	73.11%
False Positive Rate	10%
Throughput Impact	42.02%
Average Latency	0.056 seconds
95% Confidence Latency	0.123 seconds
Channel Capacity Analysis	
Theoretical Capacity	82.94 bps
Actual Capacity	22.30 bps
Bandwidth Reduction	73.11%
Anomaly Detection	
URG Pointer Without Flag	20
High Entropy URG	67
High Confidence Detections	50

5.3.2 Key Findings

- **High Detection Rate**: Achieved 73.11% detection rate with manageable false positive rate of 10%
- **Effective Bandwidth Reduction**: Successfully reduced covert channel capacity by 73.11%, from 82.94 bps to 22.30 bps
- **Primary Detection Method**: URG pointer without flag violations (20 detections) and high entropy patterns (67 detections) were the main anomaly types
- **Performance Impact**: Moderate throughput impact (42.02%) with low latency overhead (0.056s average, 0.123s 95th percentile)
- **Mitigation Efficiency**: 50 of 87 suspicious packets required active mitigation, demonstrating selective intervention
- **Real-time Capability**: Processed 119 packets in 23 seconds (5.18 packets/second) with statistical entropy analysis

5.3.3 Detection Rate Analysis

The detection rate varies inversely with bit width efficiency:

Table 8: Detection Rate by Bit Width

Bit Width	Average Detection Rate
4 bits	30.5%
8 bits	16.3%
12 bits	11.3%
16 bits	8.8%

5.3.4 Covert Channel Capacity Impact

The mitigation system achieves significant reduction in covert channel capacity:

- **4-bit channels**: 30.5% capacity reduction through detection and mitigation
- **8-bit channels**: 16.3% capacity reduction with maintained network performance
- **12-bit channels**: 11.3% capacity reduction with minimal false positives
- **16-bit channels**: 8.8% capacity reduction while preserving legitimate traffic

5.3.5 Performance Impact

The mitigation system operates with minimal performance overhead:

- **Transparent forwarding**: 100% packet forwarding success rate
- **Real-time processing**: Average processing latency under 1ms per packet
- **No false positives**: Zero interference with legitimate TCP traffic
- **Background traffic compatibility**: Seamless operation with concurrent legitimate traffic

5.4 Evaluation

5.4.1 Mitigation Effectiveness Assessment

The Phase 4 mitigation system demonstrates high effectiveness against the implemented covert channel:

Table 9: Overall Mitigation Performance

Metric	Result
Total Packets Processed	12,413
Suspicious Packets Detected	2,273
Mitigation Actions Taken	2,273
Mitigation Success Rate	100%
False Positive Rate	0%
Network Transparency	100%

5.4.2 Understanding the High Success Rate

The apparently perfect mitigation rate requires contextual understanding. According to RFC 793 [3], the TCP urgent mechanism was designed for out-of-band signaling, but RFC 6093 [4] explicitly states that "the urgent mechanism has been deprecated" due to implementation inconsistencies and limited practical utility. This deprecation creates favorable conditions for detection:

- **Rare legitimate usage**: Modern applications rarely use the URG mechanism, making any URG activity inherently suspicious
- **Protocol violations**: The implemented covert channel frequently violates RFC specifications by setting urgent pointers without URG flags
- **Controlled environment**: Testing is not done in real environment with a complex network

5.4.3 Covert Channel Evasion Strategies

The high detection rate reflects the simplicity of the implemented covert channel. More sophisticated implementations could potentially evade detection through several strategies:

- **RFC-compliant encoding**: Always set URG flag when using urgent pointer, maintaining protocol compliance while encoding data in the pointer value relative to sequence numbers
- **Mimicking legitimate traffic**: Embed covert data only in connections that simulate legitimate urgent data usage (e.g., Telnet break signals)
- **Statistical obfuscation**: Use cryptographic techniques to ensure urgent pointer values exhibit properties similar to legitimate traffic
- **Timing-based channels**: Combine urgent pointer manipulation with inter-packet timing to create hybrid covert channels
- **Sparse encoding**: Use urgent pointers infrequently and embed data across many connections to reduce detection probability
- **Dynamic adaptation**: Monitor for detection attempts and adjust encoding strategies in real-time

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.

7 Conclusion

The project demonstrated the complete lifecycle of a TCP Urgent Pointer covert channel, from implementation through detection to mitigation. Phase 2 showed that the covert channel can achieve reasonable throughput (up to 67 bps) while maintaining stealth. Phase 3 proved that RFC violation-based detection is highly effective, achieving perfect detection rates with zero false positives. Phase 4 extended this work by implementing active mitigation capabilities that successfully disrupted covert channels while maintaining complete network transparency.

The results demonstrate that while simple covert channels in the TCP Urgent Pointer field can be reliably detected due to the deprecated status of the urgent mechanism (RFC 6093), more sophisticated implementations using RFC-compliant techniques could potentially evade detection. The high success rate achieved reflects both the effectiveness of the detection approach and the simplicity of the implemented covert channel.

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.