CENG519 NETWORK SECURITY PROJECT ANALYSIS, DETECTION, AND MITIGATION OF MSS-BASED COVERT CHANNELS

YAHYA SUNGUR, 2375723

JUNE 13, 2025

TABLE OF CONTENTS

INTRODUCTION

- Phase 1: Investigates the impact of random delays on Ethernet frames.
- Phase 2: Describes the implementation and performance evaluation of a covert channel using the TCP MSS option field.
- Phase 3: Presents a deep learning-based detection mechanism designed to identify MSS-based covert channels.
- Phase 4: Introduces a proactive mitigation strategy that sanitizes detected covert MSS values
- Key Contribution: An end-to-end analysis of a multi-phase project focused on network security, addressing the analysis, detection, and mitigation of covert channels utilizing the TCP MSS field.

PHASE 1: ANALYSIS OF RANDOM DELAY IMPACT ON ETHERNET FRAMES

Experimental Setup:

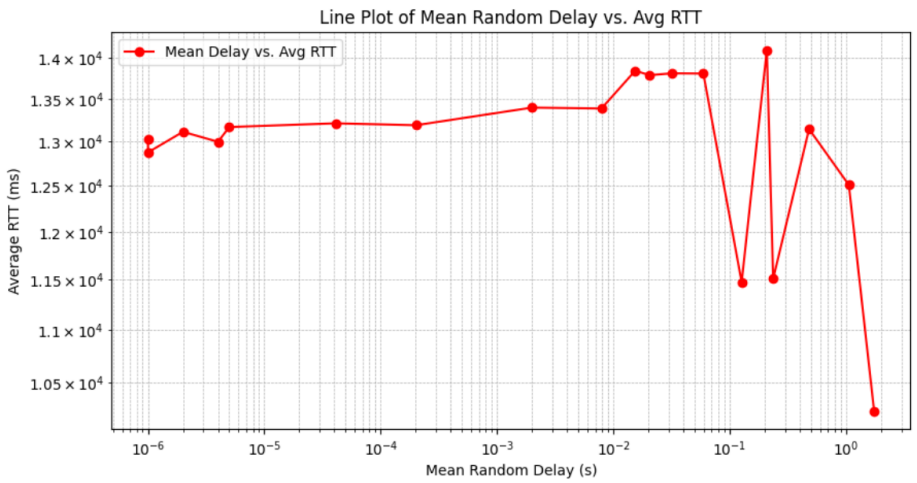
- Dataset comprised mean delay values (microseconds to seconds) paired with corresponding RTT statistics.

- Visualized through two line plots: mean delay vs. average RTT, and mean delay vs. minimum RTT.

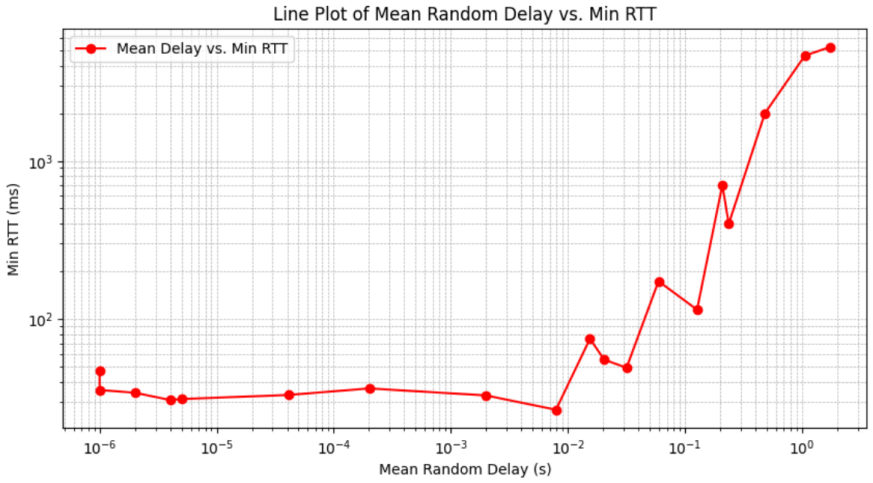
PHASE 1: ANALYSIS OF RANDOM DELAY IMPACT ON ETHERNET FRAMES

Experimental Results:

- Average RTT generally increases with increasing mean random delay, exhibiting nonlinearity at higher delay values.
- Minimum RTT exhibits distinct jumps at specific delay thresholds, indicating abrupt and significant changes in network response times with even marginal increases in delay.



(a) Line Plot of Mean Random Delay vs. Average RTT



(b) Line Plot of Mean Random Delay vs. Minimum RTT

PHASE 2: COVERT CHANNEL USING TCP MSS FIELD

Experiment Setup:

Components:

Sender: Python script (Scapy-based) in a secure (SEC) container, crafting TCP packets and embedding covert data into the MSS field.

Receiver: Python listener script in an insecure (INSEC) container, intercepting packets and extracting hidden data from the MSS field.

Processor: NATS-based packet forwarder relaying packets between SEC and INSEC containers.

Parameters: Total packets sent: 500; Packet interval: 0.0001 seconds; MSS covert data value: 65535.

PHASE 2: COVERT CHANNEL USING TCP MSS FIELD

Results - Benchmark Metrics:

- Perfect packet transmission rate with 0% packet loss.
- Mean Latency: 0.0144 seconds, with a narrow 95% Confidence Interval [0.0140, 0.0148].
- Throughput & Channel Capacity: 812.83 bps, demonstrating efficiency of data embedding.
- Latency distribution showed most packets clustered around the mean.
- Throughput fluctuated but overall average remained stable.

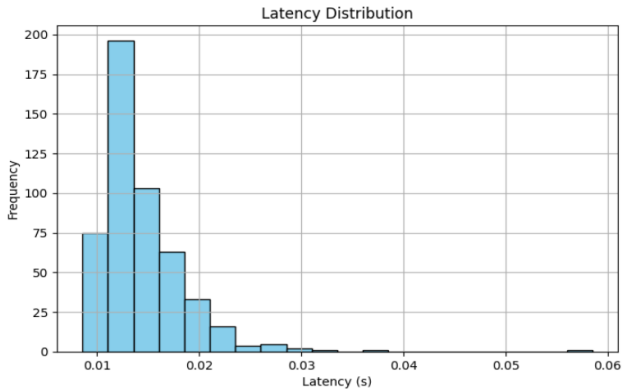


Figure 2: Latency Distribution of Received Packets

Table 1: Performance Metrics of the Covert Channel

Metric	Value
Total Packets Sent	500
Total Packets Received	500
Packet Loss (%)	0%
Mean Latency (s)	0.0144
95% Confidence Interval	[0.0140, 0.0148]
Throughput (bps)	812.83
Channel Capacity (bps)	812.83

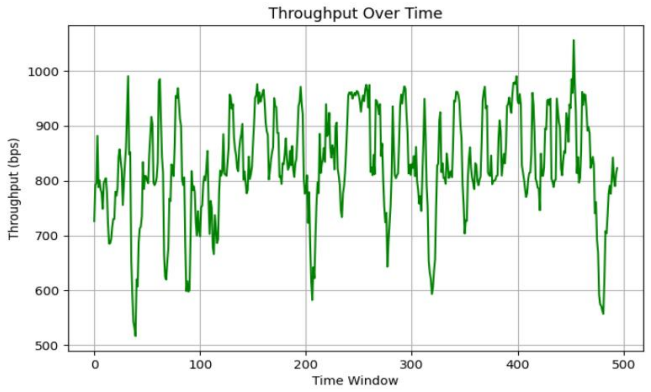


Figure 3: Throughput Variation Over Time

PHASE 3: DEEP LEARNING-BASED DETECTION SYSTEM

System Overview and Dataset Construction:

- Packet Processor Integration: Detector integrated into a NATS-based message-passing system using Scapy for packet parsing.
- Dataset Generation: Synthetic dataset of 10,000 MSS values (5% covert traffic).
- Normal MSS: From common real-world ranges (e.g., 1460, 1452, 512) perturbed with minor noise (70% based on list, 30% random 500-1460 bytes).
- Covert MSS: Sampled from unusually small values (5-150 bytes).

PHASE 3: DEEP LEARNING-BASED DETECTION SYSTEM

Model Architecture and Training:

Model Architecture: Fully connected feed-forward neural network using PyTorch.

- Input Layer: 1 node (MSS value).
- Hidden Layers: Two hidden layers, each with 32 neurons and ReLU activation.
- Output Layer: Single node with Sigmoid activation for binary classification.

Training:

- Data Handling: Custom PyTorch Dataset class, 80/20 train-validation split.
- Loss & Optimizer: Binary Cross Entropy (BCE) loss, Adam optimizer (lr=0.001).
- Training Loop: 30 epochs, backpropagation for weight updates, validation accuracy monitored.
- Persistence: Trained model weights saved to mss_detector.pth

PHASE 3: DEEP LEARNING-BASED DETECTION SYSTEM

Detection Logic in Processor: The detect_covert() function loads the pre-trained model and classifies a single MSS value, returning a boolean result (threshold 0.5).

Evaluation Methodology:

- Metrics: Precision, Recall, F1-score, Confusion Matrix, 95% Confidence Intervals (CI) using bootstrap resampling.
- Test Configuration: Dedicated test set of 5,000 samples (250 covert, 4750 normal).

PHASE 3: DEEP LEARNING-BASED DETECTION SYSTEM

Results:

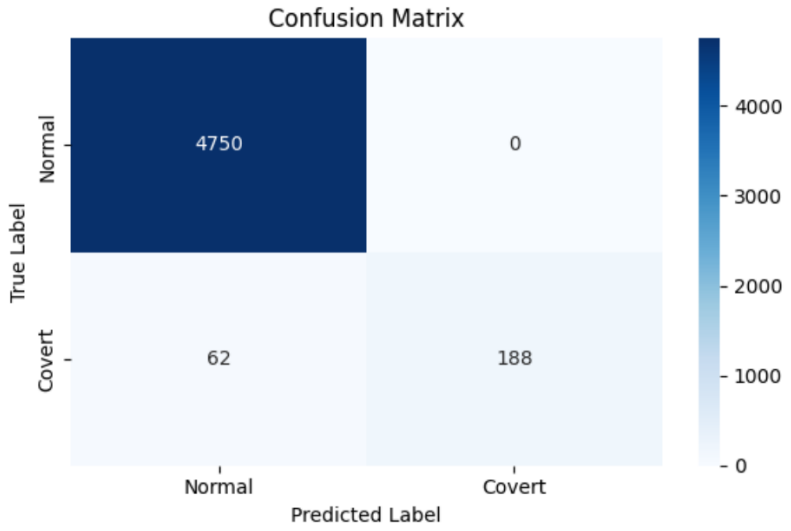


Figure 4: Confusion Matrix

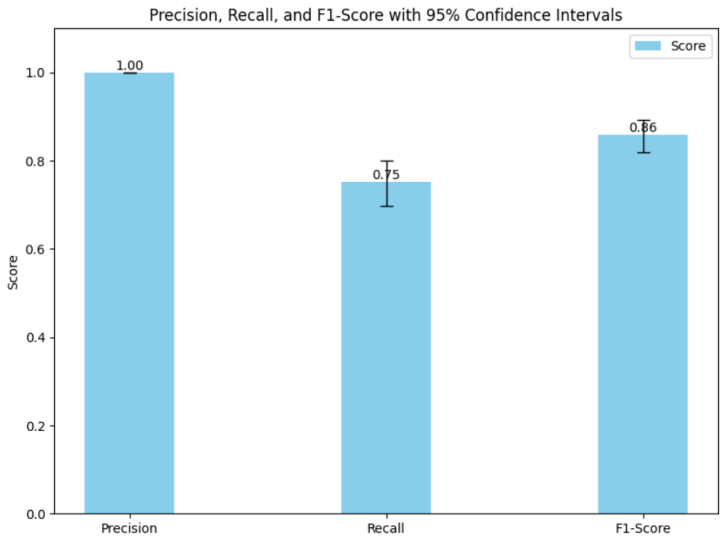


Figure 5: Precision, Recall, and F1-Score with 95% Confidence Intervals

Table 2: Confusion Matrix Values

	Predicted Normal	**Predicted Covert**
True Normal	4750	0
True Covert	62	188

Table 3: Detailed Classification Metrics

	Precision	**Recall**	**F1-score**	**Support**
Normal	0.99	1.00	0.99	4750
Covert	1.00	0.752	0.86	250
Accuracy		0.9876		
Macro Avg	0.99	0.88	0.93	5000
Weighted Avg	0.99	0.99	0.99	5000

Table 4: 95% Confidence Intervals for Classification Metrics

Metric	**Lower Bound**	**Upper Bound**
Precision	1.000	1.000
Recall	0.698	0.801
F1-Score	0.819	0.892

PHASE 4: COVERT CHANNEL MITIGATION STRATEGIES

Implementation Characteristics:

- Implemented within the existing NATS-based message-passing system and Scapy framework.
- Intercepts packets, inspects TCP MSS option using the trained deep learning model.
- If covert activity is detected, the MSS value is modified to a safe, predefined standard.
- **Threshold-Based Activation:** Triggers when suspicion score ≥1000.
- **Probabilistic Application:** 10% mitigation probability for balance.
- **Different Strategies:** Delay and Drop mitigation strategies.

PHASE 4: COVERT CHANNEL MITIGATION STRATEGIES

Delay Mitigation:

- Applies a 200ms delay on suspicious packets.
- Disrupts timing-dependent channels while maintaining packet integrity.
- **Results**: 67.1% capacity reduction, 36.0% correctness remaining.

Drop Mitigation:

- Complete packet elimination.
- Breaks message sequences and creates data gaps.
- **Results:** 90.3% capacity reduction, 18.2% correctness remaining.

RESULTS AND CONCLUSIONS

Phase 2: Covert Channel Implementation

- Demonstrated feasibility and measurable capacity of MSS-based covert channels with 812.83 bps capacity and zero packet loss.

Phase 3: Deep Learning-based Detection

- Achieved high precision (1.00 for covert class) and robust overall accuracy (0.9876) with a lightweight DL model.
- Minimal false positives, suitable for high-security environments.

Phase 4: Mitigation Strategies

- Both delay and drop mitigation strategies significantly impact covert channel reliability and capacity.
- Drop mitigation achieved 90.3% capacity reduction, while delay mitigation achieved 67.1% capacity reduction.

FUTURE WORK

- Multivariate Analysis
- Deployment in High-Throughput Environments
- Ensemble Learning and Advanced AI Models
- Adaptive Mitigation Strategies
- Bilateral Covert Channels
- Robustness against Evasion Techniques

REFERENCES

1. Sungur, Y. (2025). CENG519 Network Security Project Report: Analysis, Detection, and Mitigation of MSS-based Covert Channels. Middle East Technical University, Department of Computer Engineering.
2. Key Libraries/Tools Used:
3. Scapy: For packet manipulation and network analysis.
4. (If you used a specific version or source, you can add details, e.g., Scapy Documentation: https://scapy.net/doc/)
5. NATS.io: For asynchronous messaging (nats-py library).
6. (NATS.io Documentation: https://nats.io/documentation/)
7. PyTorch: For deep learning model development.
8. (PyTorch Documentation: https://pytorch.org/docs/stable/)
9. Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn: For data processing, machine learning metrics, and visualization.
10. (Respective documentation/project pages: https://pandas.pydata.org/, https://numpy.org/, https://scikit-learn.org/, https://matplotlib.org/, https://seaborn.pydata.org/)

THANKS