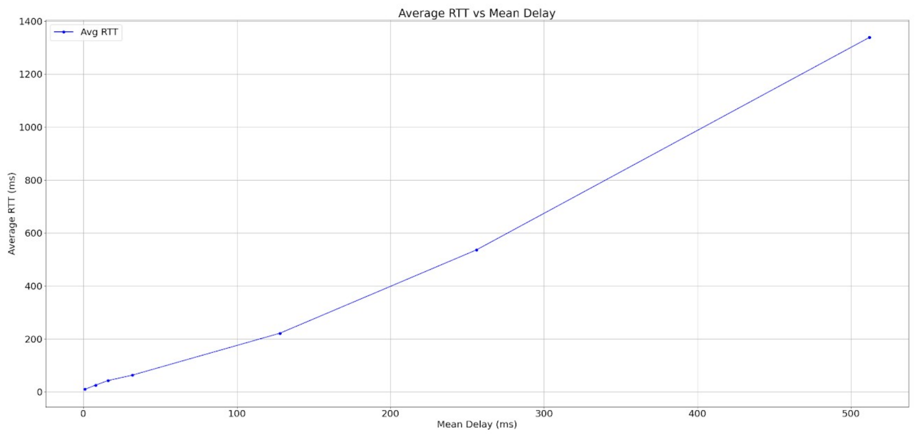
Covert Channel Project Report

Loose Source and Record Route (LSRR)

Random Packet Delays and Average RTT (Round-Trip Time)

In the first phase of our Network Security Project, we are tasked with adding random delays to package transfers between two environments, and a middle-man using a processor; then comparing the resulting mean delays with RTT (Round-Trip Time) averages. In this objective, I have chosen Python as my development language for the processor as I am more experienced in this language.

After setting up our environment, (through various tribulations such as some commands [./switch] resetting our environment without telling us). I added a pseudo randomized delay to every arriving packet in the processor. Since the pseudo-random generator I chose was using an exponential distribution with a mean that I could influence, I decided on testing our pings based on delays in ms (milliseconds), though with a floating point operation that would surely affect things. Regardless, I tested the pings with random millisecond delays of mean 1, 8, 16, 32, 128, 256 and 512. After parsing and analyzing the results, we saw a clear linear relationship between our delays and the average RTT.



Mean Delays (ms)	Average RTT (ms)
1	9.873
8	25.820
16	42.737
32	63.589
128	221.620
256	536.248
512	1338.701

As we can see in both the plot above and the table on the left. The chosen delays have a clear and linear affect on the Round-Trip Time (RTT). In fact, we can see that the RTTs almost perfectly increase with double the mean delay values, as we would expect; since a round trip is a simply a double trip with double the delays we added (as the processor is called on both ways of a ping). And it is worth mentioning that this relationship is true even though our “mean delays” have exponentially distributed randomizations and floating point uncertainties added on top of them, thus strengthening the indication of a relationship. As a result, we can clearly state that our environment is set up correctly, and the mean delays have a linear relationship with average RTTs.

Covert Channel: IPv4 Optional Header “Loose Source and Record Route (LSRR)”

In the Phase 2, we are tasked with the creation and benchmarking of a covert channel of our choice, for which I chose the Loose Source and Record Route (LSRR) in IPv4 Optional Headers as my covert channel.

LSRR is an optional header that can be used to denote a non-binding path of IPs for any packet to traverse on its way to its destination. Unlike Strict Source and Record Route (SSRR), LSRR allows the packet to diverge from any given IP if they are not-reachable, possibly resulting in only the destination IP to be used as intended.

Since the LSRR is a rarely used optional header, it is highly likely to find networks where it gets ignored instead of outright refused, as a result making it possible to send covert messages between two different networks without danger.

During the development of LSSR covert channel, we created an encoding/decoding mechanism on both sides of the connection. During the encoding stage, we encode each character into each space separated by dots, resulting in four characters per IP address in the LSRR field. And since the optional headers of a IPv4 packet has multiple bytes of free-space, we could put multiple such IP addresses inside any packet, increasing our throughput. After which, during the decoding stage, we simply decode each character separately.

Note that we used the scapy package of Python throughout this project as it allowed us to focus on encoding, decoding and benchmarking stages instead of byte manipulation through raw sockets for each packet.

But in the later stages of development, we hit the wall of optional header data limits, resulting in broken packets if we inserted more LSRR IPs than the packet can handle. After numerous trial and errors, we found the limit to be 36 character, with characters above it breaking the outgoing packet’s visible data field, possibly resulting in suspicious activity. As a result, we sent any covert messages above the 36-character limit in multiple packets in loops.

Message Length (characters)	Average Time (seconds)	95% Confidence Interval (seconds)	Capacity BPS (bits per seconds)
8	0.242462	0.019532	263.96
16	0.222935	0.013295	574.16
32	0.233931	0.013061	1094.34
64	0.466729	0.017062	1097.00
128	0.935707	0.024663	1094.36
256	1.871463	0.039767	1094.33
512	3.471792	0.053623	1179.79
1024	6.264210	0.115462	1307.75

After this set up, we made our benchmark test where we sent messages of varying sizes through a covert channel 20 times per message size. The result of which can be seen on the table below:We can see a clear linear correlation between our “character limit” and capacity bit per seconds before reaching a plateau at around 32 messages, this tracks with our 36-character limit per message that we set up, and shows the cost of sending new packages being high. We can also see that there is still a somewhat increase in capacity with larger messages, possibly from the cost of preparing the environment for different message sizes affecting the results. And lastly, we can see that average time of the messages and confidence intervals has a clear correlation with the message sizes as well. From 8, 16 and 32 character sizes each having the same average time is resultant from each one taking a single message to be sent while the rest of the sizes requiring more and more. And we can see this correlation from the average times getting doubled with each message length type, taking approximately double the packet messages to be completely sent, with a similar relation in the 95% Confidence Interval being clearly visible as well.

Loose Source and Record Route (LSRR) Covert Channel Detection

In phase 3, we are tasked with the implementation and evaluation of a **covert channel detection system** directly within the processor node of our network. For this, we re-used our set-up and covert channel protocol—specifically, IPv4 LSRR (Loose Source and Record Route) from the previous section. Then we created an experimental set-up to test the feasibility of our detection method.

For this, I have created a detector using a character-frequency-based heuristic to classify messages as covert or not, the heuristic being the real world letter frequencies of the English language. Using that heuristic, we checked how frequent each octet/letter was in a LSRR IP list, and compared it to real-world cases. Each packet was parsed in real-time using `scapy`, and were stored for data collection and analysis later on.

Benchmarking Results:

- Detection **accuracy improved with message size**, aligning with theoretical expectations that longer messages provide statistically more generalized results and are, thus, easier to classify.
- Overall detection performance yielded:

Precision	0.951
Recall	0.795
F1 Score	0.866
Accuracy	0.882

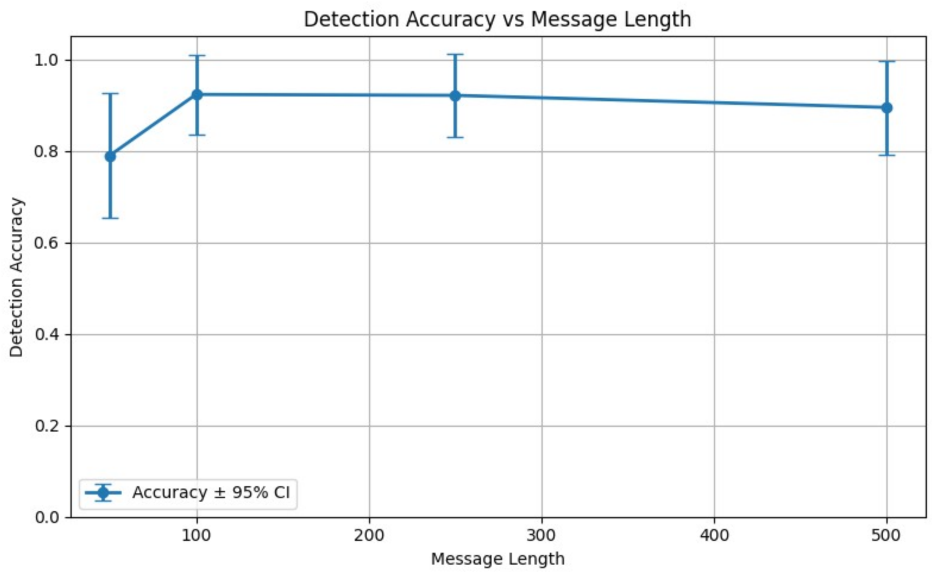
- True Positive and True Negative's made up the majority of results.

Confusion Matrix	Actual Positive	Actual Negative
Detected Positive	58	3
Detected Negative	15	77

- **95% Confidence Intervals** were computed across message sizes to quantify experimental variability.

Message Length	**Accuracy**	**95% CI**
50	0.789	± 0.136
100	0.923	± 0.088
250	0.921	± 0.090
500	0.895	± 0.102

- We can also see the correlation between message length and detection accuracy, especially the fact that messages with length of two digits wouldn't have enough statistical generalizability for the letter frequency analysis to work at its peak..



- We can also see that our algorithm doesn't result in high amounts of false positives while retaining almost 80% accuracy, concretely using message types:

Message Type	Accuracy
Covert	0.795
Visible	0.963
250	0.921
500	0.895

This phase concludes that the detection system embedded within the processor is effective and stable, particularly for message lengths above the 36-character threshold.

Loose Source and Record Route (LSRR) Covert Channel Mitigation

In Phase 4, we built upon the covert channel detection work from Phase 2 and Phase 3; aiming to mitigate the presence of covert communication via the processor node. We continued using the LSRR-based covert channel developed earlier and extended the processor's functionality to intervene in or eliminate covert channel transmission during packet processing.

To accomplish this, we employed the detector we created in the processor that targeted the packets containing the LSRR option, to find any probable covert channels and stop their packets from getting delivered past our processor.

We then ran a benchmarking study similar to that of Phase 3, sending messages of varying lengths across the covert channel, with the mitigation strategy enabled. The benchmark tested message lengths ranging from 50 to 500 characters. For each configuration, we repeated the experiments 20 times and calculated the average covert packets our scheme failed to stop.

At the end, after sending around 600 normal and 400 covert packets within 160 varying sized messages, our results in terms of caught covert messages are as follows:

Total messages	598
Failed to catch covert messages	85
Non-covert message	513
Covert message rate (mean)	0.142
Covert message rate 95% CI	0.114, 0.170

With the final channel capacity for our mitigation strategy being:

Covert messages passed	85/353, (24.08%)

Where only a quarter of total covert channels could pass. Though with the caveat that our scheme working best with large covert messages within multiple packets, while our tests had a mix of small and large sized covert messages sent. In our\ intended use-case, I theorize an even better result.

In conclusion, with Phase 4, we can see a certain degree of feasibility of our tactic of in-line covert channel mitigation via low-level packet inspection and modification. Our processor successfully neutralized the LSRR-based covert channel.

Loose Source and Record Route (LSRR) Full Mitigation

While my own mitigation strategy returns great results, there is an actually an even better way to mitigate all attacks using LSRR: **Not allowing LSRR carrying packets entirely**. This may seem like a joke or even a cop-out, but there is a real reason behind this idea: **It is already being done everywhere else**. The reality is, LSRR turned out to be a highly unsecure option among IP packets, being primarily and prominently used for DDoS attacks, and other security bypass. As a result, the Internet's own standards authority [1] and all major vendors such as Cisco [2] recommend the drop of all packets that carry LSRR with them. In other words, no-one with the right intentions would be able and willing to send LSRR carrying packets in the first place. Thus, my project on the covert channel detection and mitigation using this IPv4 option, could thankfully be only a theoretical work, in our current network environment.

References:

1) *RFC 7126: Recommendations on filtering of IPV4 packets containing IPV4 options*. (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc7126#section-4.3.5
2) *comp.dcom.sys.cisco Frequently Asked Questions (FAQ)Section - What can I do about source routing?* (n.d.). http://www.faqs.org/faqs/cisco-networking-faq/section-23.html