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.

İlker Emre Koç