CENG519: Network Security

Term Project Report

Tan Çağatay Acar

June 15, 2025

1 Covert Channel Choice

I've chosen the topic as Echo Request/Reply Payloads: Embedding data directly in ICMP echo request/reply packets. ICMP echo request/reply packets have a payload field that can be set arbitrarily. Its content and size are not defined, so we can utilize them as we see fit. However, in a confined environment with deep packet inspection precautions, this is not feasible. The system admin can confine ICMP packet payloads to the commonly used types. There are 3 payload types, one used by the ping utility in Windows, and the other 2 used by the ping utility provided by the ip-utils package included in Unix-like systems. The contents of these payloads can be seen in Table 1.

Table 1: Ping Utilities' Default Payload Patterns

Ping Utility	Payload Pattern	Payload Length
Windows	Chars abcdefghijklmnopqrstuvw (repeated)	32 bytes
Ip-utils	16-byte "timeval" + Bytes from 0x10 to 0x37	56 bytes

2 Aim

Covert channels are undesired and should be monitored and/or mitigated. Therefore, in this project, the aim is to develop an ICMP echo request/reply payload covert channel detector and mitigator, and try to beat it by maximizing the amount of covert message throughput, in order to test its performance. IBM gives a set of guidelines about detection and mitigation for covert channels that they use on their products [3]. They mandate that any covert channel higher than 0.1 bits/s should be detected, and higher than 100 bits/s should be mitigated.

3 ICMP Covert Channel

3.1 Examples From Real World

This idea of using ICMP packets for establishing covert channels is definitely not new and is probably still widely used for establishing covert channels. A sample program that uses ICMP packets for covert chatting is called ICMP-Chat. It features bidirectional chatting with automatic key exchange and encryption. One downfall of this program is it only uses reply messages. This renders the program unusable for today's internet as modern firewalls only allow reply packets if a matching request message was sent before.

Ping Tunnel is a tool for tunneling TCP traffic with ICMP. Ping Tunnel strictly uses ICMP echo request and reply packets. It directly encapsulates them without encrypting them (might be redundant if TLS is used). However, this results in abnormally large packets that can be detected easily with the right ruleset.

4 Implementation

4.1 Symmetric Request and Replies

To mimic nominal ICMP traffic, we should always work with matching echo requests and reply messages. While this is an overhead, we can use the reply messages as acknowledgments and make our communication robust. If the message reply has not been received for a timeout period, the message is resent.

4.2 Traffic Shape

Another issue is ICMP echo traffic shape is periodic. So we need to send data at some period defined by a parameter we call the interval i. This sets our message period, and we always send messages at this rate. Message size is also critical, as ICMP echo messages are usually used with payloads of 32 (Windows) or 56 (ip-utils) bytes. This limits our achievable throughput. A network monitoring admin can also limit the maximum frequency of ICMP packets sent by a source. We limit this in our mitigator to a minimum period of 0.32s.

4.3 Payload Content

We need to be careful of deep packet inspection. The messages we send will not be the same as the default payload content sent from the ping utilities. Therefore, we need to conform to the existing payload contents. The only option to embed covert information is to use ip-utils's ping utility payload.

4.3.1 The "timeval" Field

The payload has a timeval field that holds the UNIX timestamp value to be used in ping and jitter calculation. The timeval struct has two 8-byte fields called secs and usecs, which hold the timestamp's second and millisecond values. We can embed covert bits into the timeval field. If a detector is just checking the pattern field and not the timeval field, this gives us 16 bytes, or 128 bits of covert message potential.

4.3.2 The "usecs" Field

Despite being 8 bytes, the usecs field can only hold values from 0 to 999999. This forces 44 most significant bits of the field to be 0, so we cannot use them. A similar logic can limit the secs field also, but there can be a much stricter rule to limit covert information for that field that will be discussed later. For the time being, we are confined to 84 bits of covert message.

4.3.3 The "secs" Field

The timestamp stored in the timeval field has an expected behaviour that can be checked. First of all, its value should always increase for consecutive messages. Second, we can limit the maximum amount of deviation of the timestamp from the world clock that is provided by the NTP protocol. This deviation can be arbitrary, but selecting a large difference like 2048 seconds avoids the need for time zone correction calculations, while limiting the covert information to 12 bits in the secs field. This constrains the total covert information to 32 bits.

Since the traffic shape should be periodic, we can also check the difference between the timestamps of the consecutive messages from the same IP and with the same ICMP ID. We expect the difference to be close to the difference in message arrival times. We can assign a threshold to the maximum allowed deviation from these differences and detect covert information. Assigning a threshold of 1 second forbids the chance of putting any covert bits in the secs field for consecutive messages. This limits the total covert bits for the consecutive messages to 20 bits. If we select 32 ms as a threshold, then this reduces the total covert bits to 16 bits. The 16-bit capability is still hard to reach because in order to recover the covert information, the sender and the receiver should somehow have the same time value up to 1 s. This is infeasible even with microcontrollers on the same CAN network that are 1 meter apart [1]; therefore, the practical limit should be tested.

4.4 Covert Sender

A Python script is written in Sec to send covert ICMP packets to InSec. This sender can take a custom covert message to be sent as a parameter. The size parameter s denotes the covert message length in bits to be embedded in the timeval field. Another configuration parameter, u, denotes the length of covert message bits in the usecs field; the rest of the bits will be embedded in the secs field. The "r" flag is implemented so that all the covert messages after the first message are generated randomly to mimic binary information flow in tests. The "n" flag disables the acknowledgment mechanism, so the code doesn't try to resend any message. The code also measures the average achieved throughput and prints it when terminated with `Ctrl+C`.

4.5 Covert Receiver

A Python script is written in InSec to receive covert ICMP packets from Sec. This receiver takes the same s and u parameters as the sender, and they should match. The received bits are printed as they arrive.

4.6 Python-Processor

A Python script is written in Python-Processor to examine the packets between Sec and InSec. We don't allow and drop any fragmented packets to make coding easier, and the network safer, according to RFC 8900 [2]. We only inspect ICMP packets and perform the inspection independently for all source IPs. The inspection follows the techniques described in section 4. If a packet is detected as a covert message, it is marked. The detector can only mark packets once they've arrived; it cannot mark past packets even if it concludes that the packet was a covert message. The marked packages are dropped to mitigate the covert channel.

5 Recommended Demonstration

To run the setup, you need to run Python processor, then run `covert_receiver_v2.py` in InSec, and then run `covert_sender_v2.py` in Sec. Please ensure you run from bash, not sh (Virtual environment "venv" is automatically activated in this fashion). You can use python `covert_sender_v2.py -h` for different configuration parameter options.

5.1 Formal Steps

First, please run `docker compose up -d`

5.2 Python-Processor

In a terminal, run `docker compose exec -it python-processor bash -c 'python main.py'`

5.3 InSec

In another terminal, run `docker compose exec -it insec bash -c 'python covert_receiver_v2.py -s 8 -u 8'`

5.4 Sec

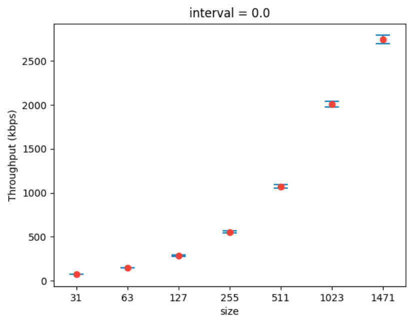
In another terminal, run `docker compose exec -it sec bash` Then you can run `python covert_sender_v2.py -s 8 -u 8 -n` for sending covert messages. To see the configurable parameters, please run `python covert_sender_v2.py -h`

6 Benchmarking

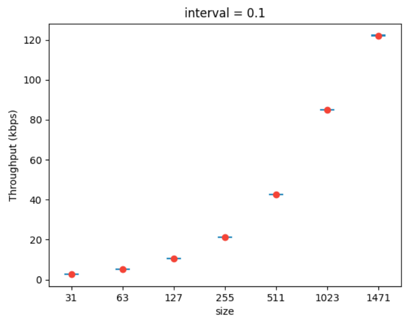
All timing results are obtained by analyzing 50 average throughput calculations for each run configuration.

6.1 ICMP Covert Channel Capability

A primitive ICMP packet sender that can utilize the full payload is tested with various packet sizes and sending intervals. Achieved throughput can be seen in Figure 1, with %95 confidence intervals visible. This is a potential test that shows the achievable covert throughput in an uncontrolled environment. A simple payload length check could make most of the results invalid.



(a) Interval = 0s



(b) Interval = 0.1s

Figure 1: ICMP covert channel capability.

We see that our throughput increases with increasing payload size and decreasing sending interval. This is a trivial result, but we can also see that we can get up to 2.8 Mbps with our covert channel. This shows the dangerous potential of the ICMP and the requirement for covert channel regulation.

6.2 ICMP Covert Channel Detection

An ICMP packet inspector and covert channel detector are implemented in Python-Processor, with the described features in section 4. The packets that are detected as covert messages are marked. The tests are done with ip-utils' ping utility for normal traffic generation and the covert sender in Sec for covert channel creation. The results can be seen in Table 2. The false negatives are high because we used configuration parameters that are known to be able to penetrate the detector. The maximum achievable bitrate is low as discussed in the following section. The high value of false positives is caused by the maximum allowed traffic frequency limit.

TP	TN	FP	FN	F_1 Score
8304	23231	6980	18382	0.396

Table 2: Detection scores

6.3 ICMP Covert Channel Mitigation

An ICMP covert channel mitigator is implemented in Python-Processor. This mitigator drops the packets that are marked by the detector. The mitigator is tested with the covert sender in Sec with various configurations. The bandwidths limited by the mitigator can be seen in Figure 2. These were tested with all the covert bits embedded in the "usecs" field. The tests show that we can limit the maximum achievable covert channel by utilizing ICMP payload to 43.02 bps. This bitrate is much smaller than the 2.8 Mbps potential, and is within the limits of IBM's guidelines.

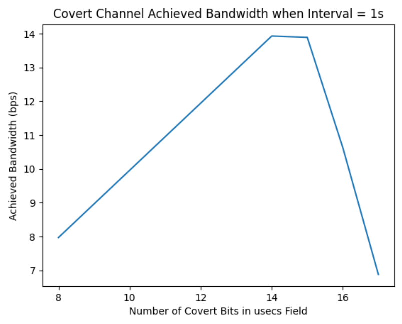
7 Discussion

We observed that we can get up to 2.8 Mbps of covert channel with the ICMP protocol if the network is not regulated. The detector we built inspects the ICMP traffic and detects covert channels that have a higher bandwidth capability than 43 bps. Covert channels with lower bandwidth than this are hard to detect, and the detector fails to do so. The mitigator with the detector can mitigate the covert channels and limit them to 43.02 bps. While this is an improvement, this can be further enhanced by doing "usecs" value characterization analysis using machine learning methods.

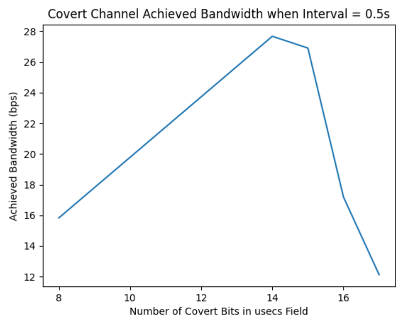
References

[1] Tan Çağatay Acar and Klaus Werner Schmidt. "Fault-Tolerant Clock Synchronization with Drift Correction for Controller Area Network". In: *2023 31st Signal Processing and Communications Applications Conference (SIU)*. 2023, pp. 1–4. DOI: 10.1109/SIU59756.2023.10223994.

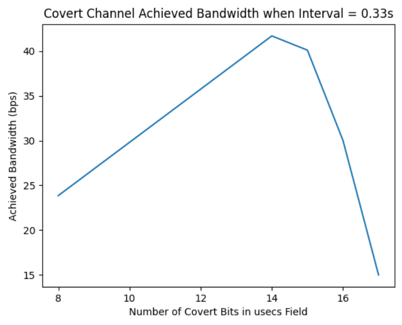
[2] Ron Bonica et al. *IP Fragmentation Considered Fragile*. RFC 8900. Sept. 2020. DOI: 10.17487/RFC8900. URL: https://www.rfc-editor.org/info/rfc8900.



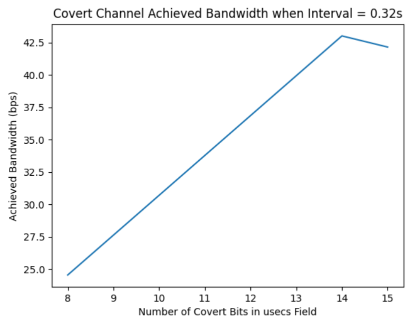
(a) Interval = 1s



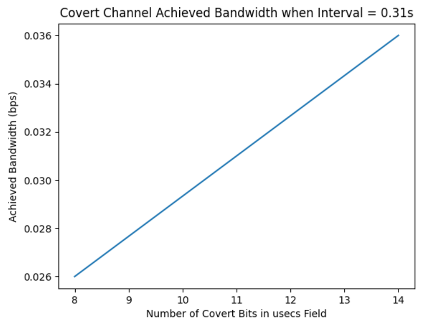
(b) Interval = 0.5s



(c) Interval = 0.33s



(d) Interval = 0.32s



(e) Interval = 0.31s

Figure 2: Achieved covert channel throughputs.

[3] IBM. *IBM AIX Product Documentation Covert Channel Bandwidth.* URL: `https://www.ibm.com/docs/en/aix/7.1.0?topic=channels-bandwidth-guidelines`.