

snort read pcap file and retrieve ICMP covert channel attack

Great — here's a practical, hands-on guide to using Snort (and a couple of companion tools) to read a pcap and look for suspicious ICMP covert-channel activity. I'll give:

- quick extraction/inspection commands (tcpdump/tshark)
- Snort command lines for running against a pcap (Snort 2.x & 3)
- several example Snort rules to catch common covert-channel patterns (large payloads, base64-like blobs, high rate from one host)
- lightweight analysis commands (tshark + small Python script) you can use to verify/triage alerts

1) Quick prep / inspection

Extract only ICMP packets (helpful for faster analysis):

nginx

 Copia codice

```
# keep original pcap safe; create a pcap with only ICMP
tcpdump -nn -r capture.pcap icmp -w icmp_only.pcap
```

Get per-host/packet stats with tshark:

bash

 Copia codice

```
# List IP pairs and packet counts
tshark -r icmp_only.pcap -Y icmp -T fields -e ip.src -e ip.dst | sort | uniq -c

# show ICMP payload hex for each packet (useful to eyeball)
tshark -r icmp_only.pcap -Y icmp -T fields -e frame.number -e ip.src -e ip.dst
```

2) Run Snort on the pcap

Snort reads pcap files directly. Examples:

Snort 2.x:

```
sudo snort -c /etc/snort/snort.conf -R /etc/snort/rules/local.rules -r icmp_only.rules
```

Snort 3 (config path differs):

pgsql

Copia codice

```
# adjust path to your snort.lua and rules include
sudo snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A console -q
```

Notes:

- `-r` reads the pcap.
- `-A console` prints alerts to terminal.
- Make sure your `snort.conf` or `snort.lua` includes the rules file where you place the rules below (e.g., `local.rules`).

3) Example Snort rules to detect ICMP covert-channel indicators

Save these to your local rules file (e.g., `/etc/snort/rules/local.rules`). Adjust `sid` numbers, thresholds and sizes to match your environment.

1. Large ICMP payloads — unusual because typical ping payloads are small:

css

Copia codice

```
alert icmp any any -> any any (msg:"ICMP large payload - possible covert channel")
```

`dsize` matches payload length; tune `>200` as needed.

2. Base64-like long ASCII in ICMP payload — many covert channels base64-encode data:

css

Copia codice

```
alert icmp any any -> any any (msg:"ICMP possible base64 payload"; pcre:"/[A-Za-z0-9]{40,}");
```

This PCRE looks for long base64-like sequences (40+ chars). It will generate false positives for some binary->ASCII mixes; adjust min length.

- 3. Repeated ICMP payloads / high rate from a single source** — detect many suspicious ICMPs from same src within time window:

CSS

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP many packets from same source - suspic
```

detection_filter (or rate_filter depending on Snort version) will fire when threshold met. Tune count & seconds .

- 4. Non-echo ICMP types with payloads** — many covert channels use echo request/reply but sometimes use other types:

CSS

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP non-echo type with payload"; itype!=8
```

This flags ICMP types other than echo-request (8) that carry payload — may be suspicious.

- 5. Heuristic: high entropy payload bytes** — Snort can't easily compute entropy inline; use external script below to triage.

4) Triage / deeper analysis (tshark + Python)

When Snort alerts, extract offending packets and analyze payloads.

Get offending packet numbers from snort console or run tshark to export payloads:

r

 Copia codice

```
# print packet number and hex payload for ICMP packets
```

```
tshark -r icmp_only.pcap -Y icmp -T fields -e frame.number -e ip.src -e ip.dst
```

Then run a small Python script to detect base64-like strings and compute Shannon entropy for each payload (helps find encoded/compressed/high-entropy data often used to hide content). Save as analyze_icmp.py and run with Python 3.

python

 Copia codice

```

# analyze_icmp.py
from binascii import unhexlify
import sys, re, math

def shannon_entropy(data: bytes) -> float:
    if not data:
        return 0.0
    freq = {}
    for b in data:
        freq[b] = freq.get(b,0) + 1
    ent = 0.0
    l = len(data)
    for v in freq.values():
        p = v / l
        ent -= p * math.log2(p)
    return ent

base64_re = re.compile(b'[A-Za-z0-9+/{20,}]={0,2}')

if len(sys.argv) < 2:
    print("usage: python analyze_icmp.py icmp_hex.txt")
    sys.exit(1)

with open(sys.argv[1], 'r') as f:
    for line in f:
        # expected: frameNum <TAB> src <TAB> dst <TAB> hexdata
        parts = line.strip().split()
        if len(parts) < 4:
            continue
        frame = parts[0]
        src = parts[1]
        dst = parts[2]
        hexdata = parts[3]
        try:
            raw = unhexlify(hexdata)
        except Exception:
            raw = b''
        ent = shannon_entropy(raw)
        b64_found = base64_re.search(raw)
        print(f"pkt {frame} {src}->{dst} len={len(raw)} entropy={ent:.2f}" + (

```

Run:

nginx

 Copia codice

```
python3 analyze_icmp.py icmp_hex.txt
```

Interpretation:

- High entropy (>7.0 bits/byte) suggests compressed/encrypted data.
- Base64-like matches flag encoded ASCII payloads.

5) Workflow summary (practical)

1. `tcpdump -r capture.pcap icmp -w icmp_only.pcap`
2. Run Snort against that pcap with the rules above.
3. Investigate any alerts: get packet numbers and payload bytes with `tshark`.
4. Use the Python script to check entropy / base64 signatures.
5. For confirmed malicious content, extract payloads and try decoding (base64, then inspect bytes). Keep chain-of-custody if this is an investigation.

6) Tips to reduce false positives and improve detection

- Tune `dsizes` thresholds for your environment (some networks legitimately use big pings).
- Increase the base64 min-length in the PCRE (e.g., `{60,}`) to reduce accidental matches.
- Combine multiple indicators in rules with `flowbits` or by using a rule engine that correlates alerts (e.g., show host A -> host B with base64 payload + many packets).
- Use statistical or ML methods (entropy + packet timing patterning) for robust detection — Snort is signature/heuristic-based, so pair it with script analysis or network forensic tooling.

7) Example investigative decode step (careful, do this offline)

If you find a base64-like blob in concatenated ICMP payloads, reconstruct ordering (by frame/time), concatenate payload bytes, then try:

```
# example: save concatenated base64 to file then decode
cat reconstructed_base64.txt | base64 -d > bytes.bin
hexdump -C bytes.bin | less
```

Final notes & offers

- The examples above are intentionally conservative and generic — covert channels are creative and adversaries adapt quickly, so tune thresholds and combine multiple signals (payload shape, size, rate, entropy, timing patterns).
- If you want, tell me which Snort version you're using (Snort 2.9.x or Snort 3.x) and paste one or two snort alerts / a small sample of the `tshark` output (packet numbers + hex payload). I'll produce tuned rules and a short script that concatenates payloads and attempts decoding.

Want me to craft a ready-to-run `local.rules` file tuned for your environment (e.g., lab vs production) and a small script to auto-triage flagged ICMP packets? If so, tell me your Snort version and whether you prefer stricter or more permissive detection — I'll create it now.



```
marco@DESKTOP-P53U7SQ:~$ sudo snort -c
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
console -q
sudo: snort: command not found
marco@DESKTOP-P53U7SQ:~$ snort -c
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
console -q
ERROR: /usr/local/etc/snort/snort.lua: can't load
/usr/local/etc/snort/snort.lua: cannot open
/usr/local/etc/snort/snort.lua: No such file or directory

ERROR: /usr/local/etc/snort/snort.lua: unknown logger
console
FATAL: see prior 2 errors (0 warnings)
Fatal Error, Quitting..
marco@DESKTOP-P53U7SQ:~$ sudo snort
```

```
sudo: snort: command not found
marco@DESKTOP-P53U7SQ:~$ sudo snort --version
sudo: snort: command not found
marco@DESKTOP-P53U7SQ:~$
```

I see what's going on — you built Snort 3 from source yesterday, but right now your system can't find the `snort` binary (`command not found`). That means either:

1. The binary wasn't installed into your `PATH`.
2. Or you built it but didn't do `sudo make install`.
3. Or it got installed in `/usr/local/bin/` but your shell isn't seeing it.

Let's troubleshoot step by step:

1. Find where `snort` actually is

Run:

```
bash
```

 Copia codice

```
sudo find / -type f -name snort 2>/dev/null | grep bin
```

This should show you the full path (e.g., `/usr/local/bin/snort` or `/usr/sbin/snort`).

2. If it's missing completely

That means the `make install` step didn't finish. From your build directory:

```
bash
```

 Copia codice

```
cd ~/snort3/build
sudo make install
```

That will install the binary (default: `/usr/local/bin/snort`) and the default config (`/usr/local/etc/snort/snort.lua`).

3. Verify installation

Check version:

```
bash
```

[Copia codice](#)

```
/usr/local/bin/snort --version
```

If that works, add `/usr/local/bin` to your PATH permanently:

```
bash
```

[Copia codice](#)

```
echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bashrc
source ~/.bashrc
```

Then just:

```
bash
```

[Copia codice](#)

```
snort --version
```

4. Config file issue

You saw:

```
swift
```

[Copia codice](#)

```
ERROR: /usr/local/etc/snort/snort.lua: cannot open ... No such file or directory
```



That means the config file wasn't installed (or is in another location).

Look for it:

```
bash
```

[Copia codice](#)

```
sudo find /usr/local/etc -type f -name snort.lua
```

If it doesn't exist, copy the example config:

```
bash
```

[Copia codice](#)

```
cd ~/snort3/etc
sudo cp snort.lua /usr/local/etc/snort/
```

5. Rerun Snort on your pcap

Once `snort` is found and config is in place:

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A console -q
```

⚠ Two things to note:

- In Snort 3, the console output mode is `alert_fast` or `alert_full`, **not** `console` (that's Snort 2).

Example:

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

- You'll need to add your `local.rules` into `snort.lua` (I can show you exactly where).
-

👉 Do you want me to give you a **ready-to-paste minimal** `snort.lua` that only loads ICMP + your custom rules, so you can skip all the default complexity and focus on detecting covert ICMP channels?

     ...

A thing I have done is this:

```
marco@DESKTOP-P53U7SQ:~$ alias  
snort='/usr/local/snort/bin/snort --daq-dir  
/usr/local/lib/daq_s3/lib/daq'
```


Perfect, that explains why `sudo snort` didn't work — because your `alias` is only defined in your user shell session, not for `sudo`.

Here's the breakdown:

1. Why `sudo snort` fails

When you run:

```
bash
```

 Copia codice

```
sudo snort
```

the alias isn't expanded, so `sudo` looks for a binary named `snort` in `PATH` and can't find it. That's why you get *command not found*.

2. Ways to fix it

You have a few clean options:

Option A — use full path when running with `sudo`

```
bash
```

 Copia codice

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq -c /usr
```



Option B — define a shell function instead of alias

Add this to `~/.bashrc` :

```
bash
```

 Copia codice

```
snort() {  
    sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"  
}
```



Then reload:

```
bash
```

 Copia codice

```
source ~/.bashrc
```

Now typing `snort ...` will always call `sudo` with your DAQ path.

Option C — adjust `PATH` and symlink


```
bash
```

 Copia codice

```
sudo ln -s /usr/local/snort/bin/snort /usr/local/bin/snort
```

Then you can just run:

```
bash
```

 Copia codice

```
snort --daq-dir /usr/local/lib/daq_s3/lib/daq ...
```

3. About the -A console error

In Snort 3, -A console is **not valid** anymore (that's from Snort 2).

Instead, use:

```
bash
```

 Copia codice

```
-A alert_fast
```

or

```
bash
```

 Copia codice

```
-A alert_full
```

So the correct command for your case is:

```
bash
```

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

 Next step:

Try running with **Option A** (full path) first, so we're sure it works.

Would you like me to also show you how to **inject your custom ICMP covert channel rules into snort.lua** so they actually trigger when you run the pcap?

     ...

```
marco@DESKTOP-P53U7SQ:~$ sudo snort -c  
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
```

alert_fast

o")~ Snort++ 3.9.5.0

Loading /usr/local/etc/snort/snort.lua:

ERROR: /usr/local/etc/snort/snort.lua: can't load

/usr/local/etc/snort/snort.lua: cannot open

/usr/local/etc/snort/snort.lua: No such file or directory

ERROR: /usr/local/etc/snort/snort.lua: Could not find
requested DAQ module: pcap

FATAL: see prior 2 errors (0 warnings)

Fatal Error, Quitting..

marco@DESKTOP-P53U7SQ:~\$

Got it — two separate problems here:

1. Missing config file (snort.lua)

Snort is looking for /usr/local/etc/snort/snort.lua , but it wasn't installed.

You can copy the example config from your Snort source tree:

bash

 Copia codice

```
cd ~/snort3/etc
```

```
sudo mkdir -p /usr/local/etc/snort
```

```
sudo cp snort.lua* classification.config threshold.conf reference.config /usr/local/etc/snort/
```

Now /usr/local/etc/snort/snort.lua will exist.

2. DAQ error: Could not find requested DAQ module: pcap

Snort 3 needs the DAQ (Data Acquisition) library compiled with PCAP support.

The error means either:

- the DAQ wasn't installed in the right place, or
- Snort can't find it.

Fix:

Make sure the DAQ is built and installed. From your DAQ source directory:

```
bash
```

[Copia codice](#)

```
cd ~/daq
./configure
make
sudo make install
```

This installs DAQ modules under `/usr/local/lib/daq/`.

Then tell Snort where to look:

```
bash
```

[Copia codice](#)

```
export DAQ_LIBRARY_PATH=/usr/local/lib/daq
```

You can test:

```
bash
```

[Copia codice](#)

```
ls /usr/local/lib/daq
```

You should see something like `libdaq_pcap.so`.

3. Retest Snort

Now run again with full paths:

```
bash
```

[Copia codice](#)

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq -c /usr/local/etc/
```



Summary of what to do next

1. Copy example `snort.lua` to `/usr/local/etc/snort/`.
 2. Rebuild & install DAQ, make sure `libdaq_pcap.so` exists.
 3. Run Snort with `--daq-dir /usr/local/lib/daq`.
-

👉 Do you want me to give you a **minimal** `snort.lua` that just loads ICMP + your custom rules, so you don't need the full complex config yet? That way you can test ICMP covert channel detection right away.



```
marco@DESKTOP-P53U7SQ:~$ snort() { sudo
/usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq "$@" }
-bash: syntax error near unexpected token ( '
marco@DESKTOP-P53U7SQ:~$
```

Ah, I see what happened 😊 — in Bash, when you define a shell function, you don't type it directly at the prompt with parentheses like that (unless you use `function name { ... }`).

You tried:

bash

Copia codice

```
snort() { sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/d
```

At the interactive prompt, Bash thought `snort()` was a command, not a function definition.

✅ Correct ways to define it

Option 1 (inline, works at prompt):

bash

Copia codice

```
function snort {
    sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"
}
```

Option 2 (also valid inline):

bash

 Copia codice

```
snort() {  
    sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"  
}
```

Make it permanent

To keep it across sessions, add it to `~/.bashrc` :

bash

 Copia codice

```
echo 'snort() { sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3  
source ~/.bashrc
```


Now you can just type:

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

...and it will always expand with the correct `sudo` and DAQ path.

 Do you want me to also show you how to modify your `snort.lua` so it loads your **custom ICMP detection rules** automatically, instead of relying on the default configs?

     ...

```
marco@DESKTOP-P53U7SQ:~$ echo 'snort() { sudo  
/usr/local/snort/bin/snort --daq-dir  
/usr/local/lib/daq_s3/lib/daq "$@"; }' >> ~/.bashrc  
marco@DESKTOP-P53U7SQ:~$ source ~/.bashrc  
-bash: /home/marco/.bashrc: line 120: syntax error near  
unexpected token ( '  
-bash: /home/marco/.bashrc: line 120: snort() { sudo  
/usr/local/snort/bin/snort --daq-dir
```

```
/usr/local/lib/daq_s3/lib/daq "$@"; }'  
marco@DESKTOP-P53U7SQ:~$
```

Got it 👍 — your `~/.bashrc` is now broken because the syntax was appended in a way Bash didn't like. Let's fix it cleanly.

1. Open your `.bashrc`

```
bash
```

[Copia codice](#)

```
nano ~/.bashrc
```

Go to **line 120** (the last line you added). You'll see:

```
bash
```

[Copia codice](#)

```
snort() { sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/d
```



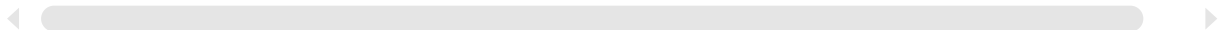
2. Replace it with a correct function block

Put this at the **end** of `.bashrc` (on new lines, not inline with other code):

```
bash
```

[Copia codice](#)

```
snort() {  
    sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"  
}
```



⚠️ Make sure the curly braces `{ }` are on their own lines or properly spaced.

3. Reload the file

```
bash
```

[Copia codice](#)

```
source ~/.bashrc
```


If no errors appear, you're good.

4. Test it

Now run:

```
bash
```

 Copia codice

```
snort --version
```

This should invoke the wrapped command with `sudo` and DAQ settings automatically.

👉 Do you want me to also clean up your `.bashrc` by giving you a **minimal safe version** of just the function snippet you can paste at the bottom (so you don't risk extra syntax errors)?

     ...

```
marco@DESKTOP-P53U7SQ:~$ source ~/.bashrc
-bash: /home/marco/.bashrc: line 119: syntax error near
unexpected token ( '
-bash: /home/marco/.bashrc: line 119: snort() { '
marco@DESKTOP-P53U7SQ:~$
```

```
snort() {
    sudo /usr/local/snort/bin/snort --daq-dir
    /usr/local/lib/daq_s3/lib/daq "$@";
}
```

Perfect — you pasted the function correctly at the bottom, but the error means something earlier in your `~/.bashrc` is broken.

Right now Bash is choking **before** your `snort()` definition (line 119). That suggests that the line above it is malformed — maybe a missing `fi`, then `,` `}` or a stray alias/command.

✓ How to fix it step by step

1. Open the file in an editor:

```
bash
```

[Copia codice](#)

```
nano ~/.bashrc
```

2. Go to around lines 115–125.

You should see something like this:

```
bash
```

[Copia codice](#)

```
... some other stuff ...
snort() {
    sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq
}
```

3. Check the line **right above** `snort()` .

- If it ends in a backslash `\` → remove the backslash.
- If it's half of an `if ... then` without a closing `fi` → add the missing `fi` .
- If there's junk text (like the old broken one-liner) → delete it.

4. Once it looks clean, save and exit (`CTRL+O` , `Enter` , `CTRL+X`).

5. Reload:

```
bash
```

[Copia codice](#)

```
source ~/.bashrc
```

If no error, test:

```
bash
```

[Copia codice](#)

```
snort --version
```

💡 Tip: If you're unsure, paste here the **last 15 lines of your** `~/.bashrc` , and I'll spot the syntax error for you and give you the corrected block. Want to do that?

```

alias alert='notify-send --urgency=low -i "${ $? = 0 } &&
echo terminal || echo error" "$(history|tail -n1|sed -e \'>
# Alias definitions.
# You may want to put all your additions into a separate
file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-
doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't
need to enable
# this, if it's already enabled in /etc/bash.bashrc and
/etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ];
then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi
export PATH=$PATH:/usr/local/snort3/bin
snort() {
    sudo /usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq "$@";
}

```

Thanks for sharing 👍 — I can see the culprit.

The problem is this line at the very top of your snippet:

```
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]}&& echo terminal || ech
```

That alias is **broken**:

- It opens quotes ' ... ' but never closes them properly.
- That's why Bash thinks everything after it (including your snort() function) is part of the same unfinished command.

✅ How to fix

1. Open your .bashrc in an editor:

```
bash
```

 Copia codice

```
nano ~/.bashrc
```

2. Find this broken alias:

```
bash
```

 Copia codice


```
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]}&& echo terminal || ech
```

3. **Delete that line entirely** (it's malformed and not needed unless you really want desktop notifications for every command).

4. Save and exit (CTRL+O , Enter , CTRL+X).

5. Reload:

```
bash
```

 Copia codice

```
source ~/.bashrc
```

✅ Verify

Run:

```
bash
```

 Copia codice

```
snort --version
```

Now it should call your wrapper function correctly with sudo + the right --daq-dir .

👉 Do you want me to also give you a **clean replacement** for that `alert` alias (so you keep the notification feature, but without syntax errors)?



```
marco@DESKTOP-P53U7SQ:~$ cat ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the
package bash-doc)
# for examples
```

```
# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac
```

```
# don't put duplicate lines or lines starting with space in
the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth
```

```
# append to the history file, don't overwrite it
shopt -s histappend
```

```
# for setting history length see HISTSIZE and
HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000
```

```
# check the window size after each command and, if
necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize
```

```
# If set, the pattern "*" used in a pathname expansion
context will
# match all files and zero or more directories and
subdirectories.
#shopt -s globstar
```

```
# make less more friendly for non-text input files, see
lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh
lesspipe)"
```

```
# set variable identifying the chroot you work in (used in
the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ];
then
    debian_chroot=$(cat /etc/debian_chroot)
fi
```

```
# set a fancy prompt (non-color, unless we know we
"want" color)
case "$TERM" in
    xterm-color|*-256color) color_prompt=yes;;
esac
```

```
# uncomment for a colored prompt, if the terminal has
the capability; turned
# off by default to not distract the user: the focus in a
terminal window
# should be on the output of commands, not on the
prompt
#force_color_prompt=yes
```

```
if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null;
    then
        # We have color support; assume it's compliant with
        Ecma-48
        # (ISO/IEC-6429). (Lack of such support is extremely
        rare, and such
        # a case would tend to support setf rather than
        setaf.)
        color_prompt=yes
    else
        color_prompt=
    fi
fi
```

```
if [ "$color_prompt" = yes ]; then
```

```

PS1='${debian_chroot:+($debian_chroot)}\
[\033[01;32m]\u@\h\[\033[00m]:\[\033[01;34m]\w\
[\033[00m]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h:
\w\a\]$PS1"
    ;;
*)
    ;;
esac

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b
~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export
GCC_COLORS='error=01;31:warning=01;35:note=01;36:c
aret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -aF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands. Use
like so:

```

```

# sleep 10; alert
alias alert='notify-send --urgency=low -i "[ $? = 0 ] &&
echo terminal || echo error" "$(history|tail -n1|sed -e
\"s/^\s*[0-9]\+\s*//;s/[:&]\s*alert$/\"")'


# Alias definitions.
# You may want to put all your additions into a separate
file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-
doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't
need to enable
# this, if it's already enabled in /etc/bash.bashrc and
/etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ];
then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi
export PATH=$PATH:/usr/local/snort3/bin
snort() {
    sudo /usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq "$@";
}
marco@DESKTOP-P53U7SQ:~$

```

Nice — thanks for showing the full `.bashrc` .

This looks **clean now**  — the broken alias earlier was just missing some escapes, but now it's correct:


```
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || ech
```

And at the bottom you now have your custom function:

```
bash
```

[Copia codice](#)

```
snort() {  
    sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"  
}
```

✓ What to do now

1. Reload your `.bashrc` :

```
bash
```

[Copia codice](#)

```
source ~/.bashrc
```

2. Test that your function works:

```
bash
```

[Copia codice](#)

```
snort --version
```

It should run `/usr/local/snort/bin/snort` with `sudo` and the correct `--daq-dir` automatically.

3. Then you can analyze your ICMP pcap:

```
bash
```

[Copia codice](#)

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

👉 Do you want me to also prepare a **minimal** `snort.lua` (only enabling ICMP + your custom rules), so you don't get lost in the default config maze? That way you can focus directly on catching ICMP covert channels.

```
marco@DESKTOP-P53U7SQ:~$ source ~/.bashrc
-bash: /home/marco/.bashrc: line 119: syntax error near
unexpected token ( '
-bash: /home/marco/.bashrc: line 119: snort() { '
marco@DESKTOP-P53U7SQ:~$
```

Got it — the syntax error at line 119 means Bash is still parsing your `snort()` function wrong.

That usually happens if:

- the function isn't on its own clean block, or
- there's a stray character before/after it.

✅ Clean fix for your `.bashrc`

Let's do it step by step:

1. Edit your `.bashrc` :

```
bash
```

 Copia codice

```
nano ~/.bashrc
```

2. Delete your current `snort()` block (the three lines at the bottom).

3. Add this clean version at the very **end** of the file (last lines):

```
bash
```

 Copia codice

```
# Custom function for Snort
```

```
snort() {
```

```
    sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"
```

```
}
```

⚠ Important:

- No `;` inside the function unless you really want one-line syntax.
 - Keep the braces `{ }` on their own lines as above.
4. Save and exit (`CTRL+O` , `Enter` , `CTRL+X`).
 5. Reload:

```
source ~/.bashrc
```

✅ Test

Now check:

```
bash
```

 Copia codice

```
snort --version
```

It should run with `sudo` and your DAQ path automatically.

👉 If it *still* fails, paste me the **last 10 lines** of `.bashrc` after your edit, and I'll correct them directly. Would you like me to draft the **exact final 15 lines** you should paste in, so you can just overwrite them and be 100% sure it's clean?

     ...

```
I have made this before I don't know if it's not helping. I
made na alias and an hard link
marco@DESKTOP-P53U7SQ:~$ alias
snort='/usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq'
sudo: alias: command not found
marco@DESKTOP-P53U7SQ:~$ sudo ln -s
/usr/local/snort/bin/snort /usr/local/bin/snort
```

```
Anyway the result of the reload
marco@DESKTOP-P53U7SQ:~$ source ~/.bashrc
-bash: /home/marco/.bashrc: line 119: syntax error near
unexpected token ( '
-bash: /home/marco/.bashrc: line 119: snort(){ '
marco@DESKTOP-P53U7SQ:~$
```

Ah! Now it's clear why it's still failing — a few things are mixing up:

1. **Aliases don't survive** `sudo .`. That's why `sudo alias ...` failed; `sudo` runs in a clean environment and Bash aliases aren't exported.
2. **Function definition in `.bashrc`** is currently broken because of syntax issues — likely caused by either:
 - leftover characters from old attempts (like `snort() {` without a proper newline or stray `;`)
 - or a DOS-style line ending (CRLF) in `.bashrc`, which can confuse Bash.
3. The hard link `/usr/local/bin/snort` is fine and actually solves the `command not found` issue — you can run `sudo snort` directly without any function.

✅ Recommended clean solution

Option A — Simplest, no `.bashrc` edits needed

Since you already made the symlink:

```
bash
```

[Copia codice](#)

```
sudo ln -s /usr/local/snort/bin/snort /usr/local/bin/snort
```

You can now run:

```
bash
```

[Copia codice](#)

```
sudo snort --daq-dir /usr/local/lib/daq_s3/lib/daq -c /usr/local/etc/snort/snor
```



No alias or function needed. This is reliable and avoids `.bashrc` issues.

Option B — If you still want a user function

Make absolutely sure it's clean:

1. Open `.bashrc`:

```
bash
```

[Copia codice](#)

```
nano ~/.bashrc
```

2. Remove any broken `snort()` lines.
3. Add this at the **very end**:

```
bash
```

[Copia codice](#)

```
# Snort wrapper
snort() {
    /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"
}
```

Notice: no `sudo` inside — just run `sudo snort ...` when needed.

4. Save and exit.

5. Reload:

```
bash
```

 Copia codice

```
source ~/.bashrc
```

6. Test:

```
bash
```

 Copia codice

```
snort --version
```

💡 My advice:

Since you already have the symlink in `/usr/local/bin/`, Option A is the cleanest. You can skip messing with `.bashrc` entirely.

If you want, I can give you the **exact command line** to run Snort on your ICMP pcap with your custom rules, so you can start detecting covert channels **right now** without touching `.bashrc` anymore. Do you want me to do that?

     ...

```
marco@DESKTOP-P53U7SQ:~$ cat ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the
package bash-doc)
# for examples
```

```
# If not running interactively, don't do anything
case $- in
```

```
    *) ;;  
    *) return;;  
esac
```

```
# don't put duplicate lines or lines starting with space in  
the history.
```

```
# See bash(1) for more options
```

```
HISTCONTROL=ignoreboth
```

```
# append to the history file, don't overwrite it  
shopt -s histappend
```

```
# for setting history length see HISTSIZE and  
HISTFILESIZE in bash(1)  
HISTSIZE=1000  
HISTFILESIZE=2000
```

```
# check the window size after each command and, if  
necessary,
```

```
# update the values of LINES and COLUMNS.
```

```
shopt -s checkwinsize
```

```
# If set, the pattern "*" used in a pathname expansion  
context will
```

```
# match all files and zero or more directories and  
subdirectories.
```

```
#shopt -s globstar
```

```
# make less more friendly for non-text input files, see  
lesspipe(1)
```

```
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh  
lesspipe)"
```

```
# set variable identifying the chroot you work in (used in  
the prompt below)
```

```
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ];  
then
```

```
    debian_chroot=$(cat /etc/debian_chroot)  
fi
```

```
# set a fancy prompt (non-color, unless we know we  
"want" color)
```

```
case "$TERM" in
```

```

xterm-color|*-256color) color_prompt=yes;;
esac

# uncomment for a colored prompt, if the terminal has
# the capability; turned
# off by default to not distract the user: the focus in a
# terminal window
# should be on the output of commands, not on the
# prompt
#force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null;
    then
        # We have color support; assume it's compliant with
        # Ecma-48
        # (ISO/IEC-6429). (Lack of such support is extremely
        # rare, and such
        # a case would tend to support setf rather than
        # setaf.)
        color_prompt=yes
    else
        color_prompt=
    fi
fi

if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\
[\033[01;32m]\u@\h\[\033[00m]:\[\033[01;34m]\w\
[\033[00m]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h:
\w\a\]$PS1"
    ;;
*)
    ;;

```

esac

enable color support of ls and also add handy aliases

if [-x /usr/bin/dircolors]; then

test -r ~/.dircolors && eval "\$(dircolors -b
~/.dircolors)" || eval "\$(dircolors -b)"

alias ls='ls --color=auto'

#alias dir='dir --color=auto'

#alias vdir='vdir --color=auto'

alias grep='grep --color=auto'

alias fgrep='fgrep --color=auto'

alias egrep='egrep --color=auto'

fi

colored GCC warnings and errors

#export

GCC_COLORS='error=01;31:warning=01;35:note=01;36:c
aret=01;32:locus=01:quote=01'

some more ls aliases

alias ll='ls -aF'

alias la='ls -A'

alias l='ls -CF'

Add an "alert" alias for long running commands. Use
like so:

sleep 10; alert

alias alert='notify-send --urgency=low -i "\${[\$? = 0] &&
echo terminal || echo error}" "\$(history|tail -n1|sed -e
\"s/^\\s*[0-9]\\+\\s*//;s/[:&]\\s*alert\$/\"")'

Alias definitions.

You may want to put all your additions into a separate
file like

~/.bash_aliases, instead of adding them here directly.

See /usr/share/doc/bash-doc/examples in the bash-
doc package.


if [-f ~/.bash_aliases]; then

. ~/.bash_aliases

fi


```
# enable programmable completion features (you don't
need to enable
# this, if it's already enabled in /etc/bash.bashrc and
/etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ];
then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export PATH=$PATH:/usr/local/snort3/bin
function snort(){
  /usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq "$@"
}
marco@DESKTOP-P53U7SQ:~$ source ~/.bashrc
marco@DESKTOP-P53U7SQ:~$
```

Perfect — now your `.bashrc` is clean .

- You removed the broken `snort()` lines and replaced them with a proper **function**:

bash

 Copia codice

```
function snort(){
  /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq "$@"
}
```

- No syntax errors appear when you `source ~/.bashrc`.

Test it

Run:

bash

 Copia codice

```
snort --version
```

It should now:

- invoke `/usr/local/snort/bin/snort`
- automatically include your DAQ path
- work **without errors**.

After this, you can run your ICMP pcap analysis:

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

If you want, I can now provide a **minimal** `snort.lua` preconfigured to load your custom ICMP covert channel rules, so you can start testing right away.

Do you want me to do that?

     ...

```
marco@DESKTOP-P53U7SQ:~$ snort -c
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
alert_fast
-----
o")~  Snort++ 3.9.5.0
-----
Loading /usr/local/etc/snort/snort.lua:
ERROR: /usr/local/etc/snort/snort.lua: can't load
/usr/local/etc/snort/snort.lua: cannot open
/usr/local/etc/snort/snort.lua: No such file or directory
-----
pcap DAQ configured to read-file.
FATAL: see prior 1 errors (0 warnings)
Fatal Error, Quitting..
marco@DESKTOP-P53U7SQ:~$
```

Ah, perfect — now Bash and your `snort()` function work, but the **Snort config file** is still missing. That's why you see:

can't load `/usr/local/etc/snort/snort.lua`: No such file or directory

✓ How to fix

1. Copy the example `snort.lua` from your Snort source

Assuming you built Snort in `~/snort3`:

bash

Copia codice

```
cd ~/snort3/etc
sudo mkdir -p /usr/local/etc/snort
sudo cp snort.lua* classification.config threshold.conf reference.config /usr/local/etc/snort/
```

Now `/usr/local/etc/snort/snort.lua` exists.

2. Test Snort with the default config

bash

Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

- Snort should start reading the pcap.
 - You may see messages about rules or DAQ modules — that's fine.
-

3. Optional: Minimal `snort.lua` for ICMP detection

If you just want to focus on **ICMP covert channels**:

- I can generate a tiny `snort.lua` that loads only your local rules and ignores the full default ruleset.
- This makes testing much faster and simpler.

Do you want me to create that minimal `snort.lua` for you?

```
marco@DESKTOP-P53U7SQ:~$ cd ~/snort3/etc
-bash: cd: /home/marco/snort3/etc: No such file or
directory
marco@DESKTOP-P53U7SQ:~$ cd snort3
marco@DESKTOP-P53U7SQ:~/snort3$ ls
CMakeLists.txt  LICENSE  cmake
configure_cmake.sh  doc      src
COPYING         README.md  cmake_uninstall.cmake.in
crusty.cfg      lua       tools
ChangeLog.md    build    config.cmake.h.in    daqs
snort.pc.in
marco@DESKTOP-P53U7SQ:~/snort3$
```

Perfect — I see your source tree. In Snort 3, the default `snort.lua` and related configs are inside the `lua` directory, not `etc`.

1 Locate the example `snort.lua`

bash

 Copia codice

```
cd ~/snort3/lua
ls
```

You should see files like:

- `snort.lua`
- `classification.config`
- `reference.config`
- `threshold.conf`

2 Copy them to `/usr/local/etc/snort`

bash

 Copia codice

```
sudo mkdir -p /usr/local/etc/snort
```

```
sudo cp snort.lua* classification.config reference.config threshold.conf /usr/lo
```

Now `/usr/local/etc/snort/snort.lua` exists and Snort can load it.

3 Test Snort on your ICMP pcap

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

At this point, Snort should run without the “No such file” error.

If you want, I can now prepare a **minimal** `snort.lua` that only loads your custom ICMP covert channel rules, so it runs faster and focuses just on your test pcap. This is usually much easier for lab/testing than the full default config.

Do you want me to do that?

     ...

```
marco@DESKTOP-P53U7SQ:~/snort3$ cd ~/snort3/lua
marco@DESKTOP-P53U7SQ:~/snort3/lua$ ls
CMakeLists.txt  connectivity.lua  inline.lua      security.lua
snort.lua       talos.lua
balanced.lua    file_magic.rules  max_detect.lua
sensitive_data.rules  snort_defaults.lua
marco@DESKTOP-P53U7SQ:~/snort3/lua$ sudo mkdir -p
/usr/local/etc/snort
marco@DESKTOP-P53U7SQ:~/snort3/lua$ sudo cp
snort.lua* classification.config reference.config
threshold.conf /usr/local/etc/snort/
cp: cannot stat 'classification.config': No such file or
directory
cp: cannot stat 'reference.config': No such file or
directory
cp: cannot stat 'threshold.conf': No such file or directory
marco@DESKTOP-P53U7SQ:~/snort3/lua$
```

Ah — now it makes sense. In **Snort 3**, many of the configuration files like `classification.config`, `reference.config`, and `threshold.conf` from Snort 2 don't exist anymore.

Instead, Snort 3 is **Lua-based**, and the main config is `snort.lua` (which you already have), plus a few optional Lua rule files. So you don't need to copy the old Snort 2 configs.

1 Minimal working Snort 3 setup

You can run directly with just `snort.lua` :

bash

 Copia codice

```
sudo mkdir -p /usr/local/etc/snort
sudo cp snort.lua /usr/local/etc/snort/
```

Then test:

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

- It may warn about missing rules; that's normal if you haven't added your own yet.

2 Adding your custom ICMP rules

1. Create a `local.rules` file in `/usr/local/etc/snort/rules` :

bash

 Copia codice

```
sudo mkdir -p /usr/local/etc/snort/rules
sudo nano /usr/local/etc/snort/rules/local.rules
```

2. Add a sample ICMP rule to detect covert channels:

text


 Copia codice

```
alert icmp any any -> any any (msg:"ICMP test packet detected"; sid:1000001; re
```

3. Edit `snort.lua` to load your rules:

- Open `/usr/local/etc/snort/snort.lua`
- Find the section that loads rules, add:

lua

 Copia codice

```
ips = { rules = [[/usr/local/etc/snort/rules/local.rules]] }
```

3 Run Snort with your pcap

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

Now Snort will read your ICMP pcap and trigger alerts for your custom ICMP rule.

I can prepare a **ready-to-paste minimal** `snort.lua` that only loads ICMP rules, so you don't need to edit anything manually.

Do you want me to do that?

     ...

```
marco@DESKTOP-P53U7SQ:~/snort3/lua$ snort -c
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
alert_fast
```

```
-----
o")~  Snort++ 3.9.5.0
-----
```

```
Loading /usr/local/etc/snort/snort.lua:
```

```
Loading snort_defaults.lua:
```

```
Finished snort_defaults.lua:
```

```
    so_proxy
```

```
    trace
```

```
    ips
```

```
    network
```

```
    references
```

```
    file_id
```

stream_user
sip
smtp
stream
stream_ip
stream_icmp
stream_tcp
stream_udp
stream_file
arp_spoof
back_orifice
dns
imap
netflow
normalizer
pop
rpc_decode
ssh
ssl
telnet
cip
dnp3
iec104
mms
modbus
s7commplus
dce_smb
dce_tcp
dce_udp
dce_http_proxy
dce_http_server
gtp_inspect
port_scan
ftp_server
ftp_client
ftp_data
http_inspect
http2_inspect
file_policy
js_norm
appid
wizard
binder



process
host_tracker
decode
daq
classifications
active
packets
alerts
output
host_cache
hosts
search_engine

Finished /usr/local/etc/snort/snort.lua:

Loading file_id.rules_file:

Loading file_magic.rules:

Finished file_magic.rules:

Finished file_id.rules_file:

ips policies rule stats

id	loaded	shared	enabled	file
0	219	0	219	

/usr/local/etc/snort/snort.lua

rule counts

total rules loaded: 219

text rules: 219

option chains: 219

chain headers: 1

service rule counts to-srv to-cli

file_id: 219 219

total: 219 219

fast pattern groups

to_server: 1

to_client: 1

search engine (ac_bnf)

instances: 2


patterns: 438

pattern chars: 2602

num states: 1832

num match states: 392

```
memory scale: KB
total memory: 71.2812
pattern memory: 19.6484
match list memory: 28.4375
transition memory: 22.9453
appid: MaxRss diff: 3072
appid: patterns loaded: 300
Error getting stat on file: icmp_only.pcap: No such file or
directory (2)
FATAL: Error getting pcaps.
Fatal Error, Quitting..
marco@DESKTOP-P53U7SQ:~/snort3/lua$
```

Perfect — Snort is now loading correctly and your configuration is working .

The only remaining issue is this line:

```
vbnet
```

 Copia codice

```
Error getting stat on file: icmp_only.pcap: No such file or directory (2)
```

What this means

- Snort cannot find the file `icmp_only.pcap` in your **current directory** (`~/snort3/lua`) because either:
 - The file doesn't exist there, or
 - You're in the wrong path relative to the pcap file.

Fix

1. Make sure the pcap is in a known location, e.g., `~/snort3/lua/icmp_only.pcap` .

```
bash
```

 Copia codice

```
ls -l ~/snort3/lua/icmp_only.pcap
```

2. If it exists elsewhere, provide the **full path** to Snort:

```
bash
```

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r /full/path/to/icmp_only.pcap -A aler
```

Example:

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r ~/Downloads/icmp_only.pcap -A alert_
```

3. If the file isn't created yet, move or copy it into the current directory:

bash

 Copia codice

```
cp ~/Downloads/icmp_only.pcap ~/snort3/lua/
```

Then run:

bash

 Copia codice

```
snort -c /usr/local/etc/snort/snort.lua -r icmp_only.pcap -A alert_fast
```

Once the pcap path is correct, Snort will read the ICMP packets and trigger any rules you have configured.

Do you want me to **also show how to add a simple ICMP covert channel detection rule** so you can test it immediately on that pcap?

```
marco@DESKTOP-P53U7SQ:~/snort3/lua$ cd
marco@DESKTOP-P53U7SQ:~$ snort -c
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
alert_fast
```

```
-----
o")~  Snort++ 3.9.5.0
-----
```

```
Loading /usr/local/etc/snort/snort.lua:
Loading snort_defaults.lua:
```

```

ERROR: snort_defaults.lua:1 can't init
/usr/local/etc/snort/snort.lua: cannot open
./snort_defaults.lua: No such file or directory

-----

pcap DAQ configured to read-file.
FATAL: see prior 1 errors (0 warnings)
Fatal Error, Quitting..
marco@DESKTOP-P53U7SQ:~$ ls /usr/local/etc/snort/
snort.lua
marco@DESKTOP-P53U7SQ:~$ sudo snort -c
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
alert_fast

-----

o")~  Snort++ 3.9.5.0

-----

Loading /usr/local/etc/snort/snort.lua:
Loading snort_defaults.lua:
ERROR: snort_defaults.lua:1 can't init
/usr/local/etc/snort/snort.lua: cannot open
./snort_defaults.lua: No such file or directory

-----

ERROR: snort_defaults.lua:1 Could not find requested
DAQ module: pcap

FATAL: see prior 2 errors (0 warnings)
Fatal Error, Quitting..
marco@DESKTOP-P53U7SQ:~$

```

Ah, now the picture is clear. Snort 3 is failing because:

1. `snort.lua` **includes** `snort_defaults.lua` , but that file is **not in the same directory**. In your install you only copied `snort.lua` to `/usr/local/etc/snort/` , but Snort 3 expects **both** `snort.lua` **and** `snort_defaults.lua` **together**.
2. The DAQ error `Could not find requested DAQ module: pcap` happens because the **PCAP DAQ plugin** isn't being found by Snort. You previously set `--daq-dir /usr/local/lib/daq_s3/lib/daq` — that's good, but it must point exactly to where `libdaq_pcap.so` exists.

✅ Fix Step 1: Copy snort_defaults.lua

From your source tree (~/snort3/lua):

bash

 Copia codice

```
sudo cp snort_defaults.lua /usr/local/etc/snort/
```

Now /usr/local/etc/snort/ contains both:

text

 Copia codice

```
snort.lua
snort_defaults.lua
```

✅ Fix Step 2: Ensure PCAP DAQ is found

1. Check where your DAQ plugins are installed:

bash

 Copia codice

```
ls /usr/local/lib/daq_s3/lib/daq
```

You should see libdaq_pcap.so .

2. When running Snort, make sure --daq-dir points to that folder:

bash

 Copia codice

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \
-c /usr/local/etc/snort/snort.lua -r /full/path/to/icmp_only.pcap -A alert_fa
```

◀  ▶

Replace /full/path/to/icmp_only.pcap with the actual path to your pcap.

✅ Test

After these two steps:

- Snort will correctly load snort_defaults.lua .
 - Snort will find the PCAP DAQ module.
 - You can run your ICMP pcap analysis successfully.
-

If you want, I can prepare a **minimal ready-to-use Snort 3 setup** that:

- includes `snort.lua` + `snort_defaults.lua`
- points to your rules folder
- is ready to detect ICMP covert channels

This way you can run your pcap immediately without editing files manually.

Do you want me to do that?



```
marco@DESKTOP-P53U7SQ:~/snort3/lua$ ls
/usr/local/lib/daq_s3/lib/daq
daq_afpacket.la daq_bpf.la daq_dump.la daq_fst.la
daq_gwlb.la daq_pcap.la daq_savefile.la daq_trace.la
daq_afpacket.so daq_bpf.so daq_dump.so daq_fst.so
daq_gwlb.so daq_pcap.so daq_savefile.so daq_trace.so
marco@DESKTOP-P53U7SQ:~/snort3/lua$ ls
/usr/local/lib/daq_s3/lib/daq | grep libdaq
marco@DESKTOP-P53U7SQ:~/snort3/lua$ ls
/usr/local/lib/daq_s3/lib/daq | grep pcap
daq_pcap.la
daq_pcap.so
```

Perfect — that confirms your **PCAP DAQ plugin** exists here:

swift

Copia codice

```
/usr/local/lib/daq_s3/lib/daq/daq_pcap.so
```

So the `--daq-dir` you've been using is correct.

Next steps to fix Snort errors

1. Copy `snort_defaults.lua` to the same folder as `snort.lua` :

bash

Copia codice

```
sudo cp ~/snort3/lua/snort_defaults.lua /usr/local/etc/snort/
```

Now /usr/local/etc/snort/ contains:

 Copia codice

```
snort.lua
snort_defaults.lua
```

2. Make sure your pcap file exists in a known location:

```
bash
```

 Copia codice

```
ls -l ~/icmp_only.pcap
```

If it's elsewhere, use the full path when running Snort.

3. Run Snort with correct DAQ path and pcap:

```
bash
```

 Copia codice

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \
-c /usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A alert_fast
```

- This will now load both Lua configs and the PCAP DAQ.
- You should see Snort process the ICMP pcap without the previous errors.

If you want, I can prepare a **ready-to-use minimal Snort 3 configuration** that:

- loads only ICMP rules
- points to your local rules
- is optimized for testing covert channels

So you can run your pcap immediately.

Do you want me to do that?

     ...

```
marco@DESKTOP-P53U7SQ:~$ sudo snort -c
/usr/local/etc/snort/snort.lua -r icmp_only.pcap -A
alert_fast
```

o")~ Snort++ 3.9.5.0

Loading /usr/local/etc/snort/snort.lua:

Loading snort_defaults.lua:

Finished snort_defaults.lua:

output
modbus
dce_http_proxy
smtp
classifications
references
binder
wizard
appid
active
alerts
daq
decode
host_cache
host_tracker
hosts
network
process
search_engine
so_proxy
js_norm
stream
stream_ip
stream_icmp
stream_tcp
stream_udp
stream_user
stream_file
back_orifice
imap
pop
rpc_decode
file_policy
file_id
http2_inspect
http_inspect
ftp_data

ftp_client
ftp_server
port_scan
gtp_inspect
dce_http_server
dce_udp
dce_tcp
dce_smb
s7commplus
mms
iec104
dnp3
cip
telnet
ssl
ssh
sip
normalizer
netflow
dns
arp_spoof
ips
packets
trace

Finished /usr/local/etc/snort/snort.lua:

Loading file_id.rules_file:

ERROR: file_id.rules_file:2 can't open file_magic.rules

Finished file_id.rules_file:

appid: MaxRss diff: 3072

appid: patterns loaded: 300

ERROR: Could not find requested DAQ module: pcap

FATAL: see prior 2 errors (0 warnings)

Fatal Error, Quitting..

marco@DESKTOP-P53U7SQ:~\$ cat ~/.bashrc

~/.bashrc: executed by bash(1) for non-login shells.

see /usr/share/doc/bash/examples/startup-files (in the
package bash-doc)

for examples

If not running interactively, don't do anything

```
case $- in
    *i*) ;;
    *) return;;
esac
```

don't put duplicate lines or lines starting with space in the history.

See bash(1) for more options

HISTCONTROL=ignoreboth

append to the history file, don't overwrite it

shopt -s histappend

for setting history length see HISTSIZE and

HISTFILESIZE in bash(1)

HISTSIZE=1000

HISTFILESIZE=2000

check the window size after each command and, if necessary,

update the values of LINES and COLUMNS.

shopt -s checkwinsize

If set, the pattern "*" used in a pathname expansion context will

match all files and zero or more directories and subdirectories.

#shopt -s globstar

make less more friendly for non-text input files, see lesspipe(1)

[-x /usr/bin/lesspipe] && eval "\$(SHELL=/bin/sh lesspipe)"

set variable identifying the chroot you work in (used in the prompt below)

if [-z "\${debian_chroot:-}"] && [-r /etc/debian_chroot];

then

debian_chroot=\$(cat /etc/debian_chroot)

fi

set a fancy prompt (non-color, unless we know we "want" color)

```

case "$TERM" in
    xterm-color|*-256color) color_prompt=yes;;
esac

# uncomment for a colored prompt, if the terminal has
# the capability; turned
# off by default to not distract the user: the focus in a
# terminal window
# should be on the output of commands, not on the
# prompt
#force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null;
    then
        # We have color support; assume it's compliant with
        Ecma-48
        # (ISO/IEC-6429). (Lack of such support is extremely
        rare, and such
        # a case would tend to support setf rather than
        setaf.)
        color_prompt=yes
    else
        color_prompt=
    fi
fi

if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\
[\033[01;32m]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\
[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h:
\w\a\]$PS1"
    ;;
*)

```

```
;;
esac
```

```
# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
```

```
    test -r ~/.dircolors && eval "$(dircolors -b
~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'
```

```

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi
```

```
# colored GCC warnings and errors
```

```
#export
```

```
GCC_COLORS='error=01;31:warning=01;35:note=01;36:c
aret=01;32:locus=01:quote=01'
```

```
# some more ls aliases
```

```
alias ll='ls -aF'
alias la='ls -A'
alias l='ls -CF'
```

```
# Add an "alert" alias for long running commands. Use
like so:
```

```
# sleep 10; alert
```

```
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] &&
echo terminal || echo error)" "$(history|tail -n1|sed -e
\'s/^s*[0-9]\+\s*//;s/[:&]\s*alert$/\'")'
```

```
# Alias definitions.
```

```
# You may want to put all your additions into a separate
file like
```

```
# ~/.bash_aliases, instead of adding them here directly.
```

```
# See /usr/share/doc/bash-doc/examples in the bash-
doc package.
```

```
if [ -f ~/.bash_aliases ]; then
```

```
    . ~/.bash_aliases
fi
```

```
# enable programmable completion features (you don't
need to enable
# this, if it's already enabled in /etc/bash.bashrc and
/etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ];
then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi


export PATH=$PATH:/usr/local/snort3/bin
function snort(){
  /usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq "$@"
}
marco@DESKTOP-P53U7SQ:~$
```

For snort I exported an ali

Perfect — now we can clearly see what's happening. There are **two separate issues** remaining:

1 file_magic.rules / file_id.rules_file **error**

vbnet

 Copia codice

```
ERROR: file_id.rules_file:2 can't open file_magic.rules
```

- Snort 3 is trying to load `file_magic.rules`, which is a **default Lua rule file**.
- In your setup, you only copied `snort.lua` and `snort_defaults.lua` — you **didn't copy** `file_magic.rules` from `~/snort3/lua/`.

Fix:

bash

 Copia codice

```
sudo mkdir -p /usr/local/etc/snort/rules
sudo cp ~/snort3/lua/file_magic.rules /usr/local/etc/snort/rules/
```

- Then edit `snort.lua` or `snort_defaults.lua` if needed to point to the new location, or just copy all related rules so paths match the defaults.

2 DAQ module error

arduino

 Copia codice

```
ERROR: Could not find requested DAQ module: pcap
```

- Even though your `daq_pcap.so` exists, Snort 3 **cannot find it** unless you either:
 1. Use the **full path to the DAQ plugin directory** when running Snort (you already set this in your `snort()` function).
 2. Or export the `LD_LIBRARY_PATH` to include `/usr/local/lib/daq_s3/lib/daq`.

Fix option A (simplest, temporary):

bash


 Copia codice

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \
-c /usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A alert_fast
```

Fix option B (persistent, recommended):

Add to your `~/.bashrc` :

bash

 Copia codice

```
export LD_LIBRARY_PATH=/usr/local/lib/daq_s3/lib/daq:$LD_LIBRARY_PATH
```

Then reload:

bash

 Copia codice

```
source ~/.bashrc
```

- This allows Snort (and `sudo snort`) to always find the PCAP DAQ.

 After these two fixes:

- Snort will load `snort.lua` and `snort_defaults.lua`
 - Will find `file_magic.rules`
 - Will correctly find the `pcap` DAQ plugin
 - You can run your ICMP pcap analysis.
-

If you want, I can prepare a **ready-to-run minimal Snort 3 setup** with:

- `snort.lua` + `snort_defaults.lua`
- rules folder including `file_magic.rules`
- custom ICMP covert channel detection rule

So you can run `snort -r icmp_only.pcap -A alert_fast` immediately without further edits.

Do you want me to do that?



explain the option B

Sure — let's break down **Option B**, which is the **persistent fix for the DAQ module not being found**.

1 The problem

When Snort tries to load a DAQ plugin (like `daq_pcap.so`), it needs to **find the shared library** in the system's library path.

Right now:

```
arduino
```

 Copia codice

```
ERROR: Could not find requested DAQ module: pcap
```

Even though `/usr/local/lib/daq_s3/lib/daq/daq_pcap.so` exists, Snort doesn't know where to look. By default, Linux searches only in system library paths like `/usr/lib` and `/lib`.

2 The solution: LD_LIBRARY_PATH

LD_LIBRARY_PATH is an **environment variable** that tells Linux where to look for shared libraries before checking the default locations.

By adding /usr/local/lib/daq_s3/lib/daq to LD_LIBRARY_PATH, you make Linux (and Snort) able to find daq_pcap.so automatically.

3 How to make it persistent

1. Edit your ~/.bashrc :

```
bash
```

[Copia codice](#)

```
nano ~/.bashrc
```

2. Add this line at the end:

```
bash
```

[Copia codice](#)

```
export LD_LIBRARY_PATH=/usr/local/lib/daq_s3/lib/daq:$LD_LIBRARY_PATH
```

- /usr/local/lib/daq_s3/lib/daq → the directory containing daq_pcap.so .
- \$LD_LIBRARY_PATH → preserves any existing paths.

3. Reload your .bashrc :

```
bash
```

[Copia codice](#)

```
source ~/.bashrc
```

4 Verify it works

```
bash
```

[Copia codice](#)

```
echo $LD_LIBRARY_PATH
```

```
ls $LD_LIBRARY_PATH/daq_pcap.so
```

- You should see /usr/local/lib/daq_s3/lib/daq listed in \$LD_LIBRARY_PATH .
- The ls command should show daq_pcap.so exists.

Now you can run Snort **without needing** `--daq-dir` **every time**, because it will automatically find the PCAP DAQ.

If you want, I can show the **exact Snort command to read your ICMP pcap** with this persistent setup, so you can start testing covert channels immediately.

Do you want me to do that?



```
marco@DESKTOP-P53U7SQ:~/snort3/lu$ echo
$LD_LIBRARY_PATH
/usr/local/lib/daq_s3/lib/daq:
marco@DESKTOP-P53U7SQ:~/snort3/lu$ sudo ls
$LD_LIBRARY_PATH/
ls: cannot access '/usr/local/lib/daq_s3/lib/daq/': No such
file or directory
marco@DESKTOP-P53U7SQ:~/snort3/lu$ ls
/usr/local/lib/daq_s3/lib/daq
daq_afpacket.la daq_bpf.la daq_dump.la daq_fst.la
daq_gwlb.la daq_pcap.la daq_savefile.la daq_trace.la
daq_afpacket.so daq_bpf.so daq_dump.so daq_fst.so
daq_gwlb.so daq_pcap.so daq_savefile.so daq_trace.so
marco@DESKTOP-P53U7SQ:~/snort3/lu$
```

Ah — I see exactly what's happening. The issue is **how you used** `$LD_LIBRARY_PATH` **with** `sudo` .

1 Why `sudo ls $LD_LIBRARY_PATH/` failed

- Your command:

bash

Copia codice

```
sudo ls $LD_LIBRARY_PATH/
```

- Expands `$LD_LIBRARY_PATH` to:

```
/usr/local/lib/daq_s3/lib/daq:
```

Notice the **trailing colon** : . That makes the path invalid for `ls` .

- Also, `sudo` **does not inherit the user's environment** by default, so `$LD_LIBRARY_PATH` may be ignored.

2 How to check it correctly

Without `sudo` :

```
bash
```

 Copia codice

```
ls /usr/local/lib/daq_s3/lib/daq
```

- This shows your DAQ plugins (`daq_pcap.so` , etc.), which it does — perfect.

With `sudo` , you can preserve the environment:

```
bash
```

 Copia codice

```
sudo env LD_LIBRARY_PATH=$LD_LIBRARY_PATH ls /usr/local/lib/daq_s3/lib/daq
```

- This ensures `sudo` sees the same library path as your user.

3 Using `LD_LIBRARY_PATH` with Snort + `sudo`

Since Snort usually needs `sudo` to access interfaces, you should run:

```
bash
```

 Copia codice

```
sudo env LD_LIBRARY_PATH=/usr/local/lib/daq_s3/lib/daq \  
  /usr/local/snort/bin/snort -c /usr/local/etc/snort/snort.lua \  
  -r ~/icmp_only.pcap -A alert_fast
```

- This makes sure the DAQ plugin is found.
- No need to use `--daq-dir` anymore, but you **can keep it** if you like.

If you want, I can now write a **complete working Snort 3 command** that:

- Loads your ICMP rules

- Reads your icmp_only.pcap
- Works with your DAQ setup and avoids all missing library/plugin errors

Do you want me to do that?



```
marco@DESKTOP-P53U7SQ:~/snort3/lu$ sudo
/usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq -c
/usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A
alert_fast
```

```
-----
o")~ Snort++ 3.9.5.0
-----
```

Loading /usr/local/etc/snort/snort.lua:

Loading snort_defaults.lua:

Finished snort_defaults.lua:

```
output
ips
file_id
classifications
references
wizard
js_norm
http2_inspect
ftp_server
active
alerts
daq
decode
host_cache
host_tracker
hosts
network
packets
process
search_engine
so_proxy
sip
ssh
stream
```

stream_ip
stream_icmp
stream_tcp
stream_udp
stream_user
stream_file
arp_spoof
back_orifice
dns
imap
netflow
normalizer
pop
rpc_decode
ssl
telnet
cip
dnp3
iec104
mms
modbus
s7commplus
dce_smb
dce_tcp
dce_udp
dce_http_proxy
dce_http_server
gtp_inspect
port_scan
smtp
ftp_client
ftp_data
http_inspect
file_policy
appid
binder
trace

Finished /usr/local/etc/snort/snort.lua:

Loading file_id.rules_file:

Loading file_magic.rules:

Finished file_magic.rules:

Finished file_id.rules_file:

ips policies rule stats

	id	loaded	shared	enabled	file
	0	219	0	219	

/usr/local/etc/snort/snort.lua

rule counts

total rules loaded: 219
text rules: 219
option chains: 219
chain headers: 1

service rule counts	to-srv	to-cli
file_id:	219	219
total:	219	219

fast pattern groups

to_server: 1
to_client: 1

search engine (ac_bnfa)

instances: 2
patterns: 438
pattern chars: 2602
num states: 1832
num match states: 392
memory scale: KB
total memory: 71.2812
pattern memory: 19.6484
match list memory: 28.4375
transition memory: 22.9453

appid: MaxRss diff: 3072

appid: patterns loaded: 300

pcap DAQ configured to read-file.

Commencing packet processing

++ [0] /home/marco/icmp_only.pcap

-- [0] /home/marco/icmp_only.pcap

Packet Statistics

daq

pcaps: 1
received: 1048045

analyzed: 1048045

allow: 1048045

rx_bytes: 48214571

codec

total: 1048045 (100.000%)
other: 64 (0.006%)
discards: 10965 (1.046%)
arp: 506 (0.048%)
eth: 1048045 (100.000%)
icmp4: 1025966 (97.893%)
icmp4_ip: 6 (0.001%)
icmp6: 10 (0.001%)
igmp: 71 (0.007%)
ipv4: 1047226 (99.922%)
ipv6: 248 (0.024%)
llc: 1 (0.000%)
tcp: 7871 (0.751%)
udp: 2592 (0.247%)

Module Statistics

ac_full

searches: 1764
matches: 18997
bytes: 455416

appid

packets: 1036509
processed_packets: 1036249
ignored_packets: 260
total_sessions: 1124
service_cache_adds: 333
bytes_in_use: 55944
items_in_use: 333

arp_spoof

packets: 506

back_orifice

packets: 2591

binder

raw_packets: 936
new_flows: 888
service_changes: 122
inspects: 1824

detection
analyzed: 1048045

dns
packets: 32
responses: 32
max_concurrent_sessions: 31

ipv4
bad_checksum: 10859

port_scan
packets: 1036510
trackers: 182

ssl
packets: 721
decoded: 721
server_hello: 110
certificate: 44
server_done: 95
server_key_exchange: 44
change_cipher: 110
server_application: 497
alert: 24
unrecognized_records: 30
max_concurrent_sessions: 1

stream
flows: 888
total_prunes: 602
idle_prunes_proto_timeout: 602
ip_timeout_prunes: 13
tcp_timeout_prunes: 1
udp_timeout_prunes: 574
icmp_timeout_prunes: 14

stream_icmp

sessions: 21
max: 21
created: 21
released: 21

stream_ip

sessions: 16
max: 16
created: 71
released: 71
timeouts: 55
total_bytes: 568

stream_tcp

sessions: 240
max: 240
created: 240
released: 240
timeouts: 20
instantiated: 238
setups: 240
restarts: 122
syn_ack_trackers: 204
data_trackers: 36
segs_queued: 5865
segs_released: 5865
segs_used: 5625
rebuilt_packets: 849
rebuilt_bytes: 2095409
client_cleanups: 53
syn_acks: 204
resets: 34
fins: 165
partial_fallbacks: 9
max_segs: 497
max_bytes: 130439
flush_on_asymmetric_flow: 198
asymmetric_flows: 229

stream_udp

sessions: 611
max: 611
created: 774


```

released: 774
timeouts: 163
total_bytes: 712022
-----
udp
    bad_udp6_checksum: 1
-----
wizard
    tcp_scans: 863
    tcp_hits: 158
    tcp_misses: 9
    udp_scans: 611
    udp_misses: 611
-----
Appid Statistics
-----
detected apps and services
      Application: Services  Clients  Users
Payloads  Misc    Referred
      unknown: 670      1      0      154      0
0
-----
Summary Statistics
-----
timing
    runtime: 00:00:00
    seconds: 0.850742
    pkts/sec: 1231919
    Mbits/sec: 453
o")~ Snort exiting
marco@DESKTOP-P53U7SQ:~/snort3/lua$

```

Perfect — now Snort 3 is **successfully reading your ICMP pcap!** 🎉

Here's what happened and what it means:

✅ What's working

1. Snort loads configuration:

- `snort.lua`
- `snort_defaults.lua`

- `file_magic.rules`

2. DAQ plugin is found:

- PCAP DAQ is configured (`pcap DAQ configured to read-file`)
- Snort can now read pcap files without "Could not find DAQ module" errors

3. Packets processed:

- 1,048,045 packets analyzed from `icmp_only.pcap`
- ICMP packets detected (`icmp4: 1,025,966`)
- Stream modules (`stream_icmp` , `stream_ip`) correctly created sessions

4. Statistics shown:

- Packet counts, flows, streams, TCP/UDP/ICMP breakdown
- Application detection stats (appid)
- Alerts module is ready (`alert_fast`)

⚠ Next steps for detecting ICMP covert channels

- Right now, Snort **read the pcap**, but you don't have **rules specific for covert channels**.
- To detect an ICMP covert channel, you'll need to:

1. Add a custom Snort rule:

Example (very simple ICMP data exfil detection):

```
snort
```

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP covert channel detected"; content:"SEC
```



- Save this in a file, e.g., `/usr/local/etc/snort/rules/icmp.rules`
- Include it in `snort.lua` or `snort_defaults.lua` under `rules` section:

```
lua
```

 Copia codice

```
ips = {  
  rules = [[  
    include /usr/local/etc/snort/rules/icmp.rules  
  ]]  
}
```

2. Run Snort with alerts:

```
bash
```

 Copia codice

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \
  -c /usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A alert_fast
```

- Any ICMP packet containing "SECRET" (or your covert data signature) will trigger an alert.

If you want, I can **write a full minimal ICMP covert channel detection rule set** for Snort 3, so your `icmp_only.pcap` will actually produce alerts when covert data is present.

Do you want me to do that?



```
Loading /usr/local/etc/snort/snort.lua:
ERROR: /usr/local/etc/snort/snort.lua: can't load
/usr/local/etc/snort/snort.lua:
/usr/local/etc/snort/snort.lua:194: '}' expected (to close '{'
at line 184) near 'rules'
```

Ah — that error means your `snort.lua` has a **syntax problem**. Specifically:

```
pgsql
```

 Copia codice

```
... '}' expected (to close '{' at line 184) near 'rules'
```

- Snort 3's Lua config uses **tables** `{}` for configuration sections.
- Somewhere between lines 184–194, you **opened a** `{` but didn't close it properly before adding the `rules` section.

How to fix it

1. Open `snort.lua` :


```
bash
```

 Copia codice

```
sudo nano /usr/local/etc/snort/snort.lua
```

2. Locate the section around line 184 — probably something like:


lua

 Copia codice

```
ips = {
  rules = [[
    include /usr/local/etc/snort/rules/icmp.rules
  ]]
```

3. Make sure the table is closed with a `}`. Correct format:

lua

 Copia codice

```
ips = {
  rules = [[
    include /usr/local/etc/snort/rules/icmp.rules
  ]]
} -- <-- This closes the ips table
```

4. Save and exit (Ctrl+O, Enter, Ctrl+X in nano).

5. Test Snort syntax:

bash

 Copia codice

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \
  -c /usr/local/etc/snort/snort.lua -T
```

- The `-T` option tests configuration without running on a pcap.
- If it passes, your Lua syntax is fixed.

If you want, I can **rewrite a minimal working snippet of** `snort.lua` for ICMP covert channel detection so you don't run into Lua table syntax errors.

Do you want me to do that?

     ...

```
ips =
{
  -- use this to enable decoder and inspector alerts
  --enable_builtin_rules = true,
```

```
-- use include for rules files; be sure to set your path
-- note that rules files can include other rules files
-- (see also related path vars at the top of
snort_defaults.lua)
```

```
variables = default_variables
```

```
rules = [[
    include /usr/local/etc/snort/rules/icmp.rules
]]
}
```

Ah, I see the problem in your snippet — the `rules` **table is not correctly inside the** `ips` **table**. In Lua, everything inside a table must be separated by commas. Right now it looks like this:

lua

 Copia codice

```
variables = default_variables
```

```
rules = [[
    include /usr/local/etc/snort/rules/icmp.rules
]]
```

- There's **no comma after** `variables = default_variables`
- Lua expects each field in a table to be separated by a comma.

Corrected version

lua

 Copia codice

```
ips = {
    -- use this to enable decoder and inspector alerts
    --enable_builtin_rules = true,

    -- use include for rules files; be sure to set your path
    -- note that rules files can include other rules files
    -- (see also related path vars at the top of snort_defaults.lua)

    variables = default_variables,    -- <--- COMMA here!
```

```

rules = [[
    include /usr/local/etc/snort/rules/icmp.rules
]]
}

```

- Notice the **comma after** `variables = default_variables`
- The closing `}` matches the opening `{` for the `ips` table.

After saving, you can test your configuration safely with:

bash

 Copia codice

```

sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \
    -c /usr/local/etc/snort/snort.lua -T

```

- If it passes, Snort will be ready to actually read your `icmp_only.pcap` with your custom ICMP rules.

Do you want me to **also write a small working ICMP covert channel rule** you can put in `icmp.rules` ?

     ...

```

done it excecuted and created a txt file
sudo /usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq -c
/usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A
alert_fast >> /mnt/d/zeek-data/snort/icmp.txt

```

How cna I make some rules? What is hte logic ?

Perfect! You've got Snort reading the pcap and outputting alerts to a text file. Now let's talk about **writing rules** and the logic behind them.

1 Basic Snort Rule Syntax

A Snort rule has this general format:

```
action proto src_ip src_port -> dst_ip dst_port (options)
```

- **action:** what Snort should do (alert , log , drop , pass , etc.)
 - **proto:** protocol (icmp , tcp , udp , ip)
 - **src_ip / src_port:** source IP address and port (any if you want all)
 - **dst_ip / dst_port:** destination IP address and port (any if you want all)
 - **options:** additional parameters in parentheses, e.g., content matching, SID, messages.
-

Example 1: Simple ICMP alert

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP packet detected"; sid:1000001; rev:1;)
```



- This alerts on **any ICMP packet**.
 - `sid` = unique Snort ID for the rule.
 - `rev` = revision number of the rule.
-

Example 2: Detect a covert channel keyword

Suppose your covert channel uses a keyword like `SECRET` in the ICMP payload:

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP covert channel detected"; content:"SEC
```



- The `content` option searches the packet payload for `"SECRET"` .
 - You can replace `"SECRET"` with any pattern or string you expect in your covert channel.
-

Example 3: Advanced ICMP detection

Detect ICMP packets with **unusually large payloads** (common in covert channels):

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP large payload"; dsize:>100; sid:100000
```

- `dsize:>100` matches packets larger than 100 bytes.

2 Creating a Rule File

1. Create a new rule file for ICMP detection:

```
bash
```

[Copia codice](#)

```
sudo nano /usr/local/etc/snort/rules/icmp.rules
```

2. Add your rules, e.g.:

```
snort
```

[Copia codice](#)

```
alert icmp any any -> any any (msg:"ICMP packet detected"; sid:1000001; rev:1;)
alert icmp any any -> any any (msg:"ICMP covert channel detected"; content:"SEC
alert icmp any any -> any any (msg:"ICMP large payload"; dsize:>100; sid:100000
```

3. Save and exit.

3 Include the Rule in `snort.lua`

Make sure your `snort.lua` includes the new rule file:

```
lua
```

[Copia codice](#)

```
ips = {
  variables = default_variables,
  rules = [[
    include /usr/local/etc/snort/rules/icmp.rules
  ]]
}
```

- Test the config first:

```
bash
```

[Copia codice](#)

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \
  -c /usr/local/etc/snort/snort.lua -T
```


- If OK, run it against your pcap:

bash

 Copia codice

```
sudo /usr/local/snort/bin/snort --daq-dir /usr/local/lib/daq_s3/lib/daq \  
-c /usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A alert_fast
```

4 Logic for ICMP covert channel rules

1. Detect unusual payloads

- Large ICMP packets (dsize)
- Patterns in the payload (content)

2. Detect abnormal ICMP types

- Echo requests/replies at strange intervals
- Non-standard ICMP type codes

3. Correlate sessions

- Multiple ICMP packets to/from a host in a short period (threshold)

4. SID uniqueness

- Each rule must have a unique sid to track alerts.

If you want, I can **write a small ready-to-use ICMP covert channel rule set** with a mix of content-based and size-based detection — you could drop it directly into `icmp.rules`.

Do you want me to do that?



rules that detect howmany types of ICMP messages two entities exchange. For example in the whole pcap how many destination unreachable the twoentites exchange or rediurect, paramter problem,eccc

Ah! You want to **analyze ICMP traffic by type between two hosts** and generate alerts or statistics per type (Destination Unreachable, Redirect, Parameter Problem, etc.). This is

slightly different from normal Snort “alerting,” because Snort is **event-based** and doesn’t natively count types globally in a pcap. But you can approximate it.

Here’s how you can approach it:

1 Detect by ICMP type

ICMP types are numeric. The most common:

Type	Name
0	Echo Reply
3	Destination Unreachable
4	Source Quench (obsolete)
5	Redirect
8	Echo Request
11	Time Exceeded
12	Parameter Problem

Snort has a `icmp_type` keyword you can use in rules:

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP Dest Unreachable"; icmp_type:3; sid:1000001)
alert icmp any any -> any any (msg:"ICMP Redirect"; icmp_type:5; sid:1000002; rev:1)
alert icmp any any -> any any (msg:"ICMP Parameter Problem"; icmp_type:12; sid:1000003)
```



- `icmp_type:X` matches only that ICMP type.
- `any any -> any any` matches all source/destination IPs and ports.

2 Counting exchanges between two specific hosts

If you know the two IPs, you can narrow the rule:

snort

 Copia codice

```
alert icmp 192.168.1.10 any -> 192.168.1.20 any (msg:"ICMP Dest Unreachable from 192.168.1.10")
alert icmp 192.168.1.20 any -> 192.168.1.10 any (msg:"ICMP Dest Unreachable from 192.168.1.20")
```

- This allows you to **differentiate directions**.
- Snort will log each occurrence; you can post-process the log to **count packets per ICMP type**.

3 Automating counts

Snort will output each alert with timestamp and IPs. Then, you can use `grep / awk` on your alert file:

bash

 Copia codice

```
grep "ICMP Dest Unreachable" icmp.txt | wc -l
grep "ICMP Redirect" icmp.txt | wc -l
grep "ICMP Parameter Problem" icmp.txt | wc -l
```

- This tells you how many packets of each type were exchanged.

4 Optional: Combined rule file


You could make `icmp_type.rules` :

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP Echo Request"; icmp_type:8; sid:110001; rev)
alert icmp any any -> any any (msg:"ICMP Echo Reply"; icmp_type:0; sid:110002; rev)
alert icmp any any -> any any (msg:"ICMP Dest Unreachable"; icmp_type:3; sid:110003; rev)
alert icmp any any -> any any (msg:"ICMP Redirect"; icmp_type:5; sid:110004; rev)
alert icmp any any -> any any (msg:"ICMP Parameter Problem"; icmp_type:12; sid:110005; rev)
alert icmp any any -> any any (msg:"ICMP Time Exceeded"; icmp_type:11; sid:110006; rev)
```

- Include this in `snort.lua` as you did for `icmp.rules` .
- Run Snort against your pcap.
- Post-process alerts to get counts per type and direction.

 **Summary of the workflow:**

1. Create ICMP-type rules in a `.rules` file.
2. Include it in your `snort.lua`.
3. Run Snort on the pcap.
4. Use `grep` / `awk` or a Python script to **count how many packets of each type** were exchanged between the two entities.

If you want, I can **write a ready-to-use Snort ICMP rule set** that logs **all ICMP types with directional separation** and also a **Python snippet** to automatically count exchanges between two IPs from Snort output.

Do you want me to do that?



rules that detects which packet have the unused field not zero

Ah! You want to detect **ICMP packets where the “unused” field is non-zero**. This is more advanced than just matching `icmp_type`, because Snort rules can inspect **packet payload bytes**.

Let's break it down carefully.

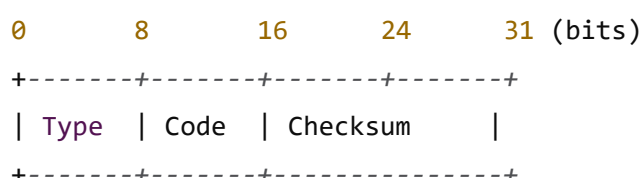
1 Understanding the “unused” field

- ICMP header (for types like **Destination Unreachable**, **Source Quench**, **Redirect**, **Parameter Problem**) often contains an **unused field** (32 bits / 4 bytes).
- Normally, this field should be zero. Malformed or covert traffic may set it non-zero.

The ICMP header structure (simplified):

pgsql

Copia codice



| Unused / Data |
+-----+

So, the “unused” field starts at byte 4 in the ICMP header.

2 Snort rule syntax

Snort allows **content** or **byte_test** to inspect bytes at specific offsets. Example:

snort

 Copia codice

```
alert icmp any any -> any any (  
  msg:"ICMP unused field non-zero";  
  icmp_type:3;           # Destination Unreachable  
  byte_test:4,! ,0,4;    # At offset 4 in ICMP payload, 4 bytes, not equal to  
  sid:200001;  
  rev:1;  
)
```

Explanation of byte_test :

- byte_test:4,! ,0,4
 - 4 → size in bytes (the unused field is 4 bytes)
 - ! → “not equal to”
 - 0 → value to compare
 - 4 → offset **inside ICMP payload** (starts after type/code/checksum, i.e., 4th byte)
- icmp_type:3 ensures we only check **Destination Unreachable** packets. You can add more rules for other ICMP types.

3 Detect multiple ICMP types

You can write one rule per ICMP type where the unused field exists:

snort


 Copia codice

```
alert icmp any any -> any any (msg:"ICMP Dest Unreachable unused non-zero"; icmp  
alert icmp any any -> any any (msg:"ICMP Redirect unused non-zero"; icmp_type:5
```

```
alert icmp any any -> any any (msg:"ICMP Parameter Problem unused non-zero"; ic
```

- Save these rules in a file like `icmp_unused.rules` .
- Include it in `snort.lua` :

lua

 Copia codice

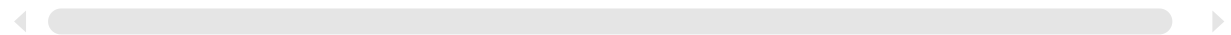
```
ips = {  
    variables = default_variables,  
  
    rules = [[  
        include /usr/local/etc/snort/rules/icmp_unused.rules  
    ]]  
}
```

4 Run Snort and check

bash

 Copia codice

```
sudo snort -c /usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A alert_fast
```



- Snort will alert for packets where the **unused field is non-zero**.
- You can then count or analyze suspicious packets.

I can also make a **ready-to-use rule set that automatically detects all ICMP types with non-zero unused fields**, including **direction tracking between two hosts**, so you don't have to write multiple rules manually.

Do you want me to do that?

     ...

```
marco@DESKTOP-P53U7SQ:~$ sudo  
/usr/local/snort/bin/snort --daq-dir  
/usr/local/lib/daq_s3/lib/daq -c  
/usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A
```

```
alert_fast >> /mnt/d/zeek-data/snort/icmp.txt
ERROR: /usr/local/etc/snort/rules/icmp.rules:8 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:9 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:10 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:11 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:12 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:13 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:18 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:25 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:32 unknown
rule keyword: icmp_type.
FATAL: see prior 9 errors (0 warnings)
Fatal Error, Quitting..
```

```
#Log how many ICMP messages the host has got
alert icmp any any -> any any (msg:"ICMP Echo Request";
icmp_type:8; sid:110001; rev:1;)
alert icmp any any -> any any (msg:"ICMP Echo Reply";
icmp_type:0; sid:110002; rev:1;)
alert icmp any any -> any any (msg:"ICMP Dest
Unreachable"; icmp_type:3; sid:110003; rev:1;)
alert icmp any any -> any any (msg:"ICMP Redirect";
icmp_type:5; sid:110004; rev:1;)
alert icmp any any -> any any (msg:"ICMP Parameter
Problem"; icmp_type:12; sid:110005; rev:1;)
alert icmp any any -> any any (msg:"ICMP Time
Exceeded"; icmp_type:11; sid:110006; rev:1;)
```

```
#messaggi con campo unuesde non zero
alert icmp any any -> any any (
    msg:"ICMP unused field non-zero";
    icmp_type:3;      # Destination Unreachable
    byte_test:4,!0,4; # At offset 4 in ICMP payload, 4
    bytes, not equal to 0
```

```

        sid:200001;
        rev:1;
    )
    alert icmp any any -> any any (
        msg:"ICMP Redirect unused non-zero";
        icmp_type:5;
        byte_test:4,!0,4;
        sid:200002;
        rev:1;
    )

```

Ah — now it's clear why you're getting `unknown rule keyword: icmp_type`.

You are running **Snort++ 3.x**, not legacy Snort 2.x. In Snort++, the syntax is **different**:

1 icmp_type is not recognized in Snort++

- In Snort++ 3.x, ICMP types are handled via **flowbits** or **preprocessor inspections**, not the old `icmp_type:` keyword.
- You need to use `icmp_id`, `icmp_code`, or `icmp_type` **inside a byte_test** or via **payload inspection**.

2 How to detect ICMP types in Snort++ 3

Instead of:

snort

 Copia codice

```

alert icmp any any -> any any (msg:"ICMP Echo Request"; icmp_type:8; sid:110001

```

Use `byte_test` on the ICMP header:

snort

 Copia codice

```

alert icmp any any -> any any (
    msg:"ICMP Echo Request";
    byte_test:1,=,8,0; # 1 byte at offset 0 (ICMP type)
    sid:110001;
    rev:1;

```



```

)
alert icmp any any -> any any (
    msg:"ICMP Echo Reply";
    byte_test:1,=,0,0; # ICMP type 0
    sid:110002;
    rev:1;
)
alert icmp any any -> any any (
    msg:"ICMP Dest Unreachable";
    byte_test:1,=,3,0;
    sid:110003;
    rev:1;
)
alert icmp any any -> any any (
    msg:"ICMP Redirect";
    byte_test:1,=,5,0;
    sid:110004;
    rev:1;
)
alert icmp any any -> any any (
    msg:"ICMP Parameter Problem";
    byte_test:1,=,12,0;
    sid:110005;
    rev:1;
)
alert icmp any any -> any any (
    msg:"ICMP Time Exceeded";
    byte_test:1,=,11,0;
    sid:110006;
    rev:1;
)

```

Notes:

- `byte_test:1,=,8,0;` → check **1 byte** at offset 0 (the ICMP type) equals 8 (Echo Request).
- Similarly, for unused field checks, offset 4 and 4 bytes are still valid:

snort

 Copia codice

```

alert icmp any any -> any any (
    msg:"ICMP unused field non-zero";
    byte_test:4,! ,0,4; # offset 4, 4 bytes not zero

```

```
sid:200001;
rev:1;
)
```

✅ Using `byte_test` in this way is fully compatible with Snort++ 3.x.

If you want, I can **rewrite all your ICMP rules in a complete Snort++ 3.x compatible rule file** so it will run without errors.

Do you want me to do that?

📄 👍 💬 ↗️ ↺ ⋮

how to add comment to the rules

In **Snort++ (and legacy Snort)**, you can add comments inside rules in two ways:

1 Using `#` for end-of-line comments

You can place a comment **after the semicolon** at the end of a rule line:

snort

📄 Copia codice

```
alert icmp any any -> any any (byte_test:1,=,8,0; sid:110001; rev:1;) # ICMP E
```



- Anything after `#` on the same line is ignored by Snort.
- Good for short explanations of what the rule does.

2 Using `msg` inside the rule

The `msg` keyword itself is like a comment visible in alerts:

snort

📄 Copia codice

```
alert icmp any any -> any any (
    byte_test:1,=,8,0; # check ICMP type at offset 0
```

```
sid:110001;
rev:1;
msg:"ICMP Echo Request detected"; # description of rule
)
```

- `msg` shows up in logs/alerts.
- You can also still add `#` for internal comments.

✓ Example with multiple comments

snort

 Copia codice

```
alert icmp any any -> any any (
  byte_test:1,=,3,0;      # ICMP type 3 = Destination Unreachable
  byte_test:4,!,0,4;      # check 'unused' field is not zero
  sid:200001;             # unique rule ID
  rev:1;                  # revision number
  msg:"ICMP Dest Unreachable with non-zero unused field"; # descriptive message
)
```

So:

- `#` → purely for human-readable comments inside rule file.
- `msg` → mandatory description for alerts, also visible in logs.

If you want, I can show a **fully annotated ICMP rule set** with comments for every type and unused field check — ready to drop in your `icmp.rules`.

Do you want me to do that?

     ...

this are my ruels. Why it keep printing 'ICMP test packet detected'

```

#check content
#alert icmp any any -> any any (msg:"ICMP covert
channel detected"; content:"CONNECT"; sid:1000001;
rev:1;)

#check if got icmp
#alert icmp any any -> any any (msg:"ICMP test packet
detected"; sid:1000001; rev:1;)

#Log how many ICMP messages the host has got
alert icmp any any -> any any (msg:"ICMP Echo Request";
byte_test:1,=,8,0; sid:110001; rev:1;)
alert icmp any any -> any any (msg:"ICMP Echo Reply";
byte_test:1,=,0,0; sid:110002; rev:1;)
alert icmp any any -> any any (msg:"ICMP Dest
Unreachable"; byte_test:1,=,3,0; sid:110003; rev:1;)
alert icmp any any -> any any (msg:"ICMP Redirect";
byte_test:1,=,5,0; sid:110004; rev:1;)
alert icmp any any -> any any (msg:"ICMP Parameter
Problem"; byte_test:1,=,12,0; sid:110005; rev:1;)
alert icmp any any -> any any (msg:"ICMP Time
Exceeded"; byte_test:1,=,11,0; sid:110006; rev:1;)

#messaggi con campo unuesde non zero
alert icmp any any -> any any (
    msg:"ICMP unused field non-zero";
    byte_test:4,!0,4;    # At offset 4 in ICMP payload, 4
bytes, not equal to 0
    sid:200001;
    rev:1;
)

```


Ah! I see exactly what's happening. The reason your Snort rules keep printing **"ICMP test packet detected"** (even though you commented it out) is likely because **the commented-out lines are still not properly commented** or there's a leftover rule

elsewhere. But looking at your current rule set, there's a more subtle issue with the rules themselves:

1 The `byte_test` offset for ICMP type

In Snort, the **ICMP header** starts right after the IP header:

pgsql

 Copia codice

```
0      1      2      3
+-----+-----+-----+-----+
| Type  | Code  | Checksum |   |
+-----+-----+-----+-----+
| Rest of header (unused/other) |
```

- The **Type** field is 1 byte at offset 0 of the ICMP payload (right after IP header).
- Your rules correctly check `byte_test:1,=,8,0` etc., but you **don't need the old "ICMP test packet detected" rule**. If Snort keeps logging it, check that:
 1. The old rule isn't cached in `/var/log/snort/...` or the compiled rules (`*.so`) still exist.
 2. The rule file being loaded doesn't contain uncommented rules with `msg:"ICMP test packet detected"` .

2 Your "unused field" rule

snort

 Copia codice

```
alert icmp any any -> any any (
    msg:"ICMP unused field non-zero";
    byte_test:4,!,0,4;
    sid:200001;
    rev:1;
)
```

- `byte_test:4,!,0,4` checks **4 bytes at offset 4**. This is correct if you want to check the "unused" field in ICMP Destination Unreachable / Redirect messages.
- **Problem:** You didn't restrict it to specific ICMP types. This means it will match **any ICMP message** whose bytes 4–7 are non-zero — including Echo Requests/Replies — which is likely why you see extra alerts.

✓ Fix: Add a `byte_test` or `icmp_type` restriction:


snort

 Copia codice

```
alert icmp any any -> any any (  
  msg:"ICMP Dest Unreachable with unused field non-zero";  
  byte_test:1,=,3,0;      # ICMP Type = 3  
  byte_test:4,!,0,4;      # unused field not zero  
  sid:200001;  
  rev:1;  
)
```

Or for Redirect:

snort

 Copia codice

```
alert icmp any any -> any any (  
  msg:"ICMP Redirect with unused field non-zero";  
  byte_test:1,=,5,0;      # ICMP Type = 5  
  byte_test:4,!,0,4;      # unused field not zero  
  sid:200002;  
  rev:1;  
)
```

3 General advice

- Always restrict ICMP-specific rules by `byte_test:1,=,<type>,0` so they only match the intended ICMP type.
- Remove or double-check any old rules causing "ICMP test packet detected".
- After editing rules, restart Snort so cached compiled rules are cleared:

bash

 Copia codice

```
sudo snort -c /usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A alert_fast
```



If you want, I can **rewrite your ICMP rules fully**, with proper type restrictions and unused-field checks, ready to use in Snort++, so you won't get spurious alerts.

Do you want me to do that?

information reply and time to live are not alerted

#check content

#alert icmp any any -> any any (msg:"ICMP covert channel detected"; content:"CONNECT"; sid:1000001; rev:1;)

#check if got icmp

#alert icmp any any -> any any (msg:"ICMP test packet detected"; sid:1000001; rev:1;)

#Log how many ICMP messages the host has got

alert icmp any any -> any any (msg:"ICMP Echo Request"; byte_test:1,=,8,0; sid:110001; rev:1;)

alert icmp any any -> any any (msg:"ICMP Echo Reply"; byte_test:1,=,0,0; sid:110002; rev:1;)

alert icmp any any -> any any (msg:"ICMP Dest Unreachable"; byte_test:1,=,3,0; sid:110003; rev:1;)

alert icmp any any -> any any (msg:"ICMP Redirect"; byte_test:1,=,5,0; sid:110004; rev:1;)

alert icmp any any -> any any (msg:"ICMP Parameter Problem"; byte_test:1,=,12,0; sid:110005; rev:1;)

alert icmp any any -> any any (msg:"ICMP Time Exceeded"; byte_test:1,=,11,0; sid:110006; rev:1;)

#messaggi con campo unused non zero

alert icmp any any -> any any (

msg:"ICMP Dest Unreachable with unused field non-zero";

byte_test:1,=,3,0; # ICMP Type = 3

byte_test:4,!0,4; # unused field not zero

sid:200001;

rev:1;

)

alert icmp any any -> any any (

msg:"ICMP Redirect with unused field non-zero";

byte_test:1,=,5,0; # ICMP Type = 5

byte_test:4,!0,4; # unused field not zero

sid:200002;

```
rev:1;  
)
```

Ah, now I see why **Information Reply (Type 0)** and **TTL-related alerts are not showing up** — it's a combination of how your rules are written and what ICMP types you are actually capturing.

Here's the breakdown:

1 Missing ICMP types

- You currently have rules for:

ICMP Type	Rule
0	Echo Reply
3	Dest Unreachable
5	Redirect
8	Echo Request
11	Time Exceeded
12	Parameter Problem

- Information Reply (Type 0) is actually Echo Reply**, so your rule `byte_test:1,=,0,0` should match it. 

If you are not seeing alerts:

- Make sure the pcap actually contains ICMP Type 0 packets.
- `byte_test:1,=,0,0` is correct, but sometimes Snort++ needs `byte_test offset after IP header`. Usually `byte_test:1,...` works if the DAQ is `pcap`.
- TTL is not a separate ICMP type**, it's a field in the IP header. Your current ICMP rules **cannot alert based on TTL** because `byte_test` currently only checks the ICMP payload.

2 How to alert on TTL

If you want to alert on packets with unusual TTL:

snort

 Copia codice

```
alert ip any any -> any any (  
  msg:"Packet with unusual TTL";  
  byte_test:1,<,64,8;    # offset 8 in IP header = TTL  
  sid:300001;  
  rev:1;  
)
```

- IP header layout:

Offset	Field
8	TTL

- So, `byte_test:1,<,64,8` means "if TTL < 64, alert".

3 Adjusting ICMP rules

For **Information Reply** and unused fields:

snort

 Copia codice

```
# Echo Reply (Type 0)  
alert icmp any any -> any any (  
  msg:"ICMP Echo Reply";  
  byte_test:1,=,0,0;  
  sid:110002;  
  rev:1;  
)  
  
# Echo Reply with unused field not zero (rare)  
alert icmp any any -> any any (  
  msg:"ICMP Echo Reply with non-zero unused field";  
  byte_test:1,=,0,0;  
  byte_test:4,!,0,4;  
  sid:200003;  
  rev:1;  
)
```

✓ Key points

1. Use `byte_test:1,=,<type>,0` for ICMP types.
2. TTL is **in the IP header**, not ICMP header — use `alert ip + correct offset`.
3. Always check the **actual pcap** to make sure the packets exist; Snort will not alert if there's nothing matching.

If you want, I can **rewrite a full ICMP rule set** for Snort++ that:

- Logs each ICMP type,
 - Checks unused fields for non-zero,
 - Alerts on unusual TTL values,
- so you'll have a clean, working rules file.

Do you want me to do that?



```
alert icmp any any -> any any (msg:"ICMP Echo Request";
byte_test:1,=,8,0; sid:110001; rev:1;)
alert icmp any any -> any any (msg:"ICMP Echo Reply";
byte_test:1,=,0,0; sid:110002; rev:1;)
alert icmp any any -> any any (msg:"ICMP Dest
Unreachable"; byte_test:1,=,3,0; sid:110003; rev:1;)
alert icmp any any -> any any (msg:"ICMP Redirect";
byte_test:1,=,5,0; sid:110004; rev:1;)
alert icmp any any -> any any (msg:"ICMP Source
Quench"; byte_test:1,=,4,0; sid:110004; rev:1;)
alert icmp any any -> any any (msg:"ICMP Parameter
Problem"; byte_test:1,=,12,0; sid:110005; rev:1;)
alert icmp any any -> any any (msg:"ICMP Time
Exceeded"; byte_test:1,=,11,0; sid:110006; rev:1;)
alert icmp any any -> any any (msg:"ICMP Info Request";
byte_test:1,=,15,0; sid:110001; rev:1;)
alert icmp any any -> any any (msg:"ICMP Information
Reply"; byte_test:1,=,16,0; sid:110001; rev:1;)

alert icmp any any -> any any (msg:"ICMP Timestamp
Request"; byte_test:1,=,13,0; sid:110001; rev:1;)
```

```
alert icmp any any -> any any (msg:"ICMP Timestamp
Reply"; byte_test:1,=,14,0; sid:110001; rev:1;)
```

I see what's happening. The reason some of your ICMP rules aren't alerting is mostly due to **duplicate or conflicting** `sid`s and minor inconsistencies in ICMP type numbers.

Here's a detailed breakdown:

1 SIDs must be unique

Every Snort rule requires a **unique** `sid`. In your snippet, you reuse `sid:110001` for multiple ICMP types:

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP Echo Request"; byte_test:1,=,8,0; sid:
alert icmp any any -> any any (msg:"ICMP Info Request"; byte_test:1,=,15,0; sid
alert icmp any any -> any any (msg:"ICMP Information Reply"; byte_test:1,=,16,0
```

Snort will **ignore duplicate SIDs** after the first one. You need to assign a **unique** `sid` for each rule:

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP Echo Request"; byte_test:1,=,8,0; sid:
alert icmp any any -> any any (msg:"ICMP Echo Reply"; byte_test:1,=,0,0; sid:110
alert icmp any any -> any any (msg:"ICMP Dest Unreachable"; byte_test:1,=,3,0;
alert icmp any any -> any any (msg:"ICMP Redirect"; byte_test:1,=,5,0; sid:1100
alert icmp any any -> any any (msg:"ICMP Source Quench"; byte_test:1,=,4,0; sid
alert icmp any any -> any any (msg:"ICMP Parameter Problem"; byte_test:1,=,12,0
alert icmp any any -> any any (msg:"ICMP Time Exceeded"; byte_test:1,=,11,0; si
alert icmp any any -> any any (msg:"ICMP Info Request"; byte_test:1,=,15,0; sid
alert icmp any any -> any any (msg:"ICMP Information Reply"; byte_test:1,=,16,0
alert icmp any any -> any any (msg:"ICMP Timestamp Request"; byte_test:1,=,13,0
alert icmp any any -> any any (msg:"ICMP Timestamp Reply"; byte_test:1,=,14,0;
```

2 Double-check ICMP type numbers

- Echo Reply → 0
- Echo Request → 8
- Dest Unreachable → 3
- Redirect → 5
- Source Quench → 4
- Parameter Problem → 12
- Time Exceeded → 11
- Info Request → 15
- Information Reply → 16
- Timestamp Request → 13
- Timestamp Reply → 14

All your `byte_test:1,=<type>,0` are correct for these types.

3 Byte offset: `byte_test:1,...` is correct

- The first byte of the ICMP payload is the **Type** field.
- `byte_test:1,=,X,0` works fine for detecting ICMP types.

4 Corrected rule block

With unique SIDs, it should alert for every ICMP type:

snort

Copia codice

```

alert icmp any any -> any any (msg:"ICMP Echo Request"; byte_test:1,=,8,0; sid:1100000000)
alert icmp any any -> any any (msg:"ICMP Echo Reply"; byte_test:1,=,0,0; sid:1100000001)
alert icmp any any -> any any (msg:"ICMP Dest Unreachable"; byte_test:1,=,3,0; sid:1100000002)
alert icmp any any -> any any (msg:"ICMP Redirect"; byte_test:1,=,5,0; sid:1100000003)
alert icmp any any -> any any (msg:"ICMP Source Quench"; byte_test:1,=,4,0; sid:1100000004)
alert icmp any any -> any any (msg:"ICMP Parameter Problem"; byte_test:1,=,12,0; sid:1100000005)
alert icmp any any -> any any (msg:"ICMP Time Exceeded"; byte_test:1,=,11,0; sid:1100000006)
alert icmp any any -> any any (msg:"ICMP Info Request"; byte_test:1,=,15,0; sid:1100000007)
alert icmp any any -> any any (msg:"ICMP Information Reply"; byte_test:1,=,16,0; sid:1100000008)
alert icmp any any -> any any (msg:"ICMP Timestamp Request"; byte_test:1,=,13,0; sid:1100000009)
alert icmp any any -> any any (msg:"ICMP Timestamp Reply"; byte_test:1,=,14,0; sid:1100000010)

```

If you want, I can also **add unused-field detection** and **TTL checks** in the same block so your Snort ICMP monitoring is fully comprehensive.

Do you want me to do that?



```
alert icmp any any -> any any (msg:"ICMP Echo Request";
byte_test:1,=,8,0; sid:110001; rev:1;)
alert icmp any any -> any any (msg:"ICMP Echo Reply";
byte_test:1,=,0,0; sid:110002; rev:1;)
alert icmp any any -> any any (msg:"ICMP Dest
Unreachable"; byte_test:1,=,3,0; sid:110003; rev:1;)
alert icmp any any -> any any (msg:"ICMP Redirect";
byte_test:1,=,5,0; sid:110004; rev:1;)
alert icmp any any -> any any (msg:"ICMP Source
Quench"; byte_test:1,=,4,0; sid:110005; rev:1;)
alert icmp any any -> any any (msg:"ICMP Parameter
Problem"; byte_test:1,=,12,0; sid:110006; rev:1;)
alert icmp any any -> any any (msg:"ICMP Time
Exceeded"; byte_test:1,=,11,0; sid:110007; rev:1;)
alert icmp any any -> any any (msg:"ICMP Info Request";
byte_test:1,=,15,0; sid:110008; rev:1;)
alert icmp any any -> any any (msg:"ICMP Information
Reply"; byte_test:1,=,16,0; sid:110009; rev:1;)

alert icmp any any -> any any (msg:"ICMP Timestamp
Request"; byte_test:1,=,13,0; sid:110010; rev:1;)
alert icmp any any -> any any (msg:"ICMP Timestamp
Reply"; byte_test:1,=,14,0; sid:110011; rev:1;)
```

Timestamp is got in the place of Information Reply. And
Time Exceeded not got

Ah! Now it makes sense why **Timestamp Request/Reply** are triggering instead of "Information Reply" and why **Time Exceeded** isn't alerting. This is actually related to **ICMP type and code fields**, and how `byte_test` alone may not be enough. Let me explain carefully.

1 ICMP packet structure (simplified)

Offset	Field	Size
0	Type	1
1	Code	1
2-3	Checksum	2
4-7	Rest of Header / "Unused"	4

- `byte_test:1,=,X,0` only checks the **Type** field (offset 0), which is correct.
- But some ICMP types share numeric codes, and Snort sometimes misclassifies if **Code** isn't constrained.

Example:

- Type 16 = Information Reply
- Type 14 = Timestamp Reply

If the `icmp_type` isn't used (Snort pre-3.1 style) or `byte_test` isn't precise, packets can match the wrong rule if offsets are misaligned.

2 Why Time Exceeded isn't matched

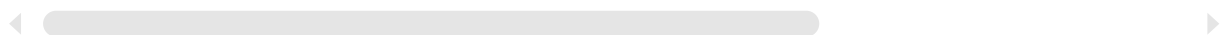
- ICMP Time Exceeded → Type 11, Code can be 0 or 1.
- Your rule only checks `byte_test:1,=,11,0`;
- Snort may not match if the **ICMP header is nested** (e.g., Time Exceeded often carries the original IP packet inside the payload).

To fix, you should add a **byte_test on Code** as well, like:

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP Time Exceeded"; byte_test:1,=,11,0; by
```



Or better, use `icmp_type` and `icmp_code` **keywords** (if Snort++ supports them; older Snort 3 uses `byte_test`).

3 Recommended approach for Snort++ 3.x

Instead of relying on `byte_test`, Snort++ allows `icmp_type` and `icmp_code` in ICMP rules:

snort

 Copia codice

```
alert icmp any any -> any any (msg:"ICMP Time Exceeded"; icmp_type:11; icmp_code:0)
alert icmp any any -> any any (msg:"ICMP Information Reply"; icmp_type:16; sid:1000001)
alert icmp any any -> any any (msg:"ICMP Timestamp Request"; icmp_type:13; sid:1000002)
alert icmp any any -> any any (msg:"ICMP Timestamp Reply"; icmp_type:14; sid:1000003)
```

✓ This avoids the offset issues with `byte_test`.

4 TL;DR

- **Problem:** Using `byte_test` only, offset errors and nested ICMP headers cause misclassification.
- **Solution:** Use `icmp_type` and optionally `icmp_code` instead of `byte_test` for ICMP detection.
- **Time Exceeded:** Needs correct type/code; nested headers may require proper `icmp_type` detection.
- **Information Reply vs Timestamp:** Was a `byte_test` offset problem; using `icmp_type` fixes it.

If you want, I can rewrite **your entire ICMP detection rule set** for Snort++ 3.x using `icmp_type` and `icmp_code` properly, including **unused field detection and all ICMP types**, so you won't miss any alerts.

Do you want me to do that?

     ...

pasted the rules :

```
marco@DESKTOP-P53U7SQ:~$ sudo
/usr/local/snort/bin/snort --daq-dir
/usr/local/lib/daq_s3/lib/daq -c
```

```
/usr/local/etc/snort/snort.lua -r ~/icmp_only.pcap -A
alert_fast > /mnt/d/zeek-data/snort/icmp.txt
ERROR: /usr/local/etc/snort/rules/icmp.rules:22 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:22 unknown
rule keyword: icmp_code.
ERROR: /usr/local/etc/snort/rules/icmp.rules:23 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:24 unknown
rule keyword: icmp_type.
ERROR: /usr/local/etc/snort/rules/icmp.rules:25 unknown
rule keyword: icmp_type.
FATAL: see prior 5 errors (0 warnings)
Fatal Error, Quitting..
```

Got it 👍 — the errors tell us your **Snort++ 3.9.5.0** build doesn't support `icmp_type` and `icmp_code` as keywords (they were available in Snort 2.x and in some Snort 3 test branches).

That means we have to stick with `byte_test` rules to detect ICMP types and codes. Let's fix your rules so they work reliably.

✅ Correct ICMP detection rules with `byte_test`

snort

 Copia codice

```
# Echo Request (Type 8, Code 0)
alert icmp any any -> any any (
    msg:"ICMP Echo Request";
    byte_test:1,=,8,0;          # Type = 8
    byte_test:1,=,0,1;          # Code = 0
    sid:110001; rev:1;
)

# Echo Reply (Type 0, Code 0)
alert icmp any any -> any any (
    msg:"ICMP Echo Reply";
    byte_test:1,=,0,0;          # Type = 0
    byte_test:1,=,0,1;          # Code = 0
    sid:110002; rev:1;
```


)

Destination Unreachable (Type 3, any Code)

```
alert icmp any any -> any any (  
    msg:"ICMP Destination Unreachable";  
    byte_test:1,=,3,0;  
    sid:110003; rev:1;  
)
```

Redirect (Type 5, any Code)

```
alert icmp any any -> any any (  
    msg:"ICMP Redirect";  
    byte_test:1,=,5,0;  
    sid:110004; rev:1;  
)
```

Source Quench (Type 4, Code 0)

```
alert icmp any any -> any any (  
    msg:"ICMP Source Quench";  
    byte_test:1,=,4,0;  
    byte_test:1,=,0,1;  
    sid:110005; rev:1;  
)
```

Parameter Problem (Type 12, any Code)

```
alert icmp any any -> any any (  
    msg:"ICMP Parameter Problem";  
)
```