New issue

Jump to bottom

Memory leak in ndpi_free_flow() #994

Closed FitzBC opened this issue on Aug 26, 2020 · 5 comments

Assignees

FitzBC commented on Aug 26, 2020 • edited

I am using `nfstream` to detect traffic, and the `dissect` option is turned on, which means that nfstream will use `nDPI` to complete this task. But I encountered a **memory leak** problem during use. Nfstream is developed based on the python language, memory leaks are rarely encountered, so I started to locate the cause.

In nfstream, I located the reason for this line of code : plugin.py. In fact, the code calls the `ndpi_flow_free()` function in nDPI: ndpi_main.c. By comparing the memory release operation in ndpi_flow_free() and the prototype of the ndpi_flow_struct structure, I think that ndpi_flow_free() has not completely released all the variables in ndpi_flow_struct.

I think this may be the root cause of the memory leak.

aouinizied commented on Aug 26, 2020 Collaborator

@FitzBC Hi, can you please provide the version of nfstream you use. Which interpreter (PyPy or CPython) and its version. How did you confirm that it's a memory leak? It was on live or offline mode?

BR,
Zied

lucaderi assigned **aouinizied** on Aug 26, 2020

FitzBC commented on Aug 26, 2020 • edited Author

@aouinizied Thanks for your reply, I am using offline mode, and my relevant version is as follows:

- nfstream: 5.2.0
- CPython: 3.7.0

For the confirm of a memory leak, first, during the running of my program, the memory usage rate keeps rising and eventually is forced to end. You can use the following simple test code to verify(Because it is a single process, the memory usage rate increases slowly, about 2 **MB/s**. You can use the process pool to speed up this process):

```
from nfstream import NFStreamer
import os

def nfs_judge_process(pcap_name, return_dict=None):

    my_awesome_streamer = NFStreamer(source=pcap_name,  # or network interface
                                     snaplen=1,
                                     idle_timeout=0,
                                     active_timeout=0,
                                     # plugins=(),
                                     dissect=True,
                                     max_tcp_dissections=1,
                                     max_udp_dissections=1,
                                     statistics=False,
                                     enable_guess=True,
                                     decode_tunnels=True,
                                     bpf_filter=None,
                                     promisc=False
                                     )

    data_pandas = my_awesome_streamer.to_pandas(ip_anonymization=False)
    application_name = data_pandas[0:1]['application_name'][0]

    return application_name

def main():

    pcap_dir='your_pcap_dir'
    while(1):
        for path, dir_list, file_list in os.walk(pcap_dir):
            file_list.sort()
            for pcap_name in file_list:

                nfs_judge_process(os.path.join(pcap_dir, pcap_name))


if __name__ == "__main__":
    main()
```

Second, I used muppy to help locate. My code is as follows:

```
from pympler import muppy, summary

all_objects_1 = muppy.get_objects()
sum1 = summary.summarize(all_objects_1)

nfs_judge_process(os.path.join(pcap_dir, pcap_name))

sum2 = summary.summarize(muppy.get_objects())
```

```
diff = summary.get_diff(sum1, sum2)
summary.print_(diff)
```

The output of each loop is similar to the following output:

```
                               types |   # objects |   total size
==================================== | =========== | ============
                                list |        7757 |      1.60 MB
                                dict |        1431 |    793.80 KB
                                 str |       10037 |    760.77 KB
                               tuple |        1141 |     94.01 KB
                                code |         545 |     76.77 KB
                                type |          77 |     76.48 KB
                                 int |        2478 |     67.79 KB
                                 set |          37 |     21.09 KB
                          re.Pattern |          25 |     20.81 KB
  pycparser.ply.yacc.MiniProduction |         310 |     16.95 KB
                 _cffi_backend.CType |         127 |     16.16 KB
                             weakref |         165 |     12.89 KB
                              method |         206 |     12.88 KB
                    weakref.KeyedRef |         132 |     11.34 KB
                   member_descriptor |         142 |      9.98 KB
```

aouinizied commented on Aug 26, 2020 • edited

Collaborator

@FitzBC Thanks for the information.

- Do you confirm the "leak" disappears when dissect is set to False?
- With your current parameters (idle=0 and active=0), each packet will be treated as a single flow. which explain why memory grows is a fast fashion. Is that to answer a specific need?

FitzBC commented on Aug 26, 2020 • edited

Author

@aouinizied

- When dissect is set to False, the leak disappeared, so I think the cause may be on the nDPI side.
- In my needs, I need to judge each packet. These packets may have different sources, so I used idle=0 and active=0.

In addition, I think that even if each packet has a single flow, if each flow will release the memory normally after it is used, it will not cause the memory to rise rapidly.

Finally, I found an interesting thing. In the nfs_judge_process() function of the sample code I provided, if you delete the following two lines and replace them with `del`, then the memory rise will become very rapid, I think this information may be helpful to you.

```
# data_pandas = my_awesome_streamer.to_pandas(ip_anonymization=False)
# application_name = data_pandas[0:1]['application_name'][0]
del my_awesome_streamer
```

aouinizied commented on Aug 27, 2020

Collaborator

@FitzBC Thank you for the provided details.
After investigation, this issue has nothing to deal with nDPI.

- Memory was not leaking. What you observed was specific to your extreme settings (idle=0, active=0) and how backpressure was handled in nfstream in such a scenario. I fixed it and the fix will be integrated into the next release of nfstream.
- Your second observation is due to another reason that is independent of the current issue. In your loop, you instantiate an NFStreamer object and you release it with the del without any processing. The issue is that some allocated modules are not correctly freed when no processing is done. This is also fixed and will be integrated into the next release.

BR,
Zied

aouinizied closed this as completed on Aug 27, 2020

Assignees

aouinizied

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants