Closed · Issue created 2 years ago by HAPPY

# Infinite memory allocation while parsing this tcp packet.

## Summary

In Wireshark 3.2.7 and the older version, the Wireshark would run into dead circle and consume infinite memory upon open the attached pcap file. This was addressed in `epan/dissectors/packet-fbzero.c` by calculating the tag offset incorrectly.

This can be exploit it by sending this special tcp packet to the remote computer. When the remote open any version of wireshark to analysing the dumped pacp file, the wireshark caused its computer exhausted with no available memory.

## Steps to reproduce

Open the `poc.pcap` in the attachment with the latest wireshark (3.2.7).

## What is the current bug behavior?

Upon open the `poc.pcap`, the wireshark has no response. (Denied Of Service)

In detail, Wireshark will run into dead circle and malloc memory continuously until the os use up all available memory.

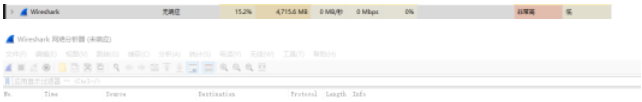
## What is the expected correct behavior?

The wireshark should parse this tcp pcap packet properly and has little memory consumption.

## Sample capture file

poc.pcap

## Relevant logs and/or screenshots



dump analysis in windbg.

```
STACK_TEXT:
000000e7`38ef6f30 00007ff9`6ebe1415 ucrtbase!__crt_stdio_output::output_processor<char,__crt_stdio_output::string_output_ad
000000e7`38ef6f70 00007ff9`6ebe1c24 ucrtbase!common_vsprintf<__crt_stdio_output::format_validation_base,char>+0x104
000000e7`38ef74a0 00007ff9`6ebe27f1 ucrtbase!__stdio_common_vsprintf_s+0x31
000000e7`38ef74e0 00007ff9`01c8dd8b libwireshark!wmem_strdup_vprintf+0x8b
000000e7`38ef7520 00007ff9`01c8dced libwireshark!wmem_strdup_printf+0x1d
000000e7`38ef7550 00007ff9`00bece41 libwireshark!val_to_str+0x31
000000e7`38ef7580 00007ff9`00f580a3 libwireshark!dissect_e212_utf8_imsi+0x559b3
000000e7`38ef7630 00007ff9`00f58419 libwireshark!dissect_e212_utf8_imsi+0x55d29
000000e7`38ef76b0 00007ff9`00f57a40 libwireshark!dissect_e212_utf8_imsi+0x55350
000000e7`38ef7700 00007ff9`00f57b18 libwireshark!dissect_e212_utf8_imsi+0x55428
000000e7`38ef7740 00007ff9`00ba3041 libwireshark!dissector_try_heuristic+0x141
000000e7`38ef77d0 00007ff9`0158ee38 libwireshark!decode_tcp_ports+0x2f8
000000e7`38ef7860 00007ff9`01596bec libwireshark!get_tcp_stream_count+0x19ac
000000e7`38ef7a60 00007ff9`0158f517 libwireshark!decode_tcp_ports+0x9d7
000000e7`38ef7b10 00007ff9`01591546 libwireshark!dissect_tcp_payload+0xe6
000000e7`38ef7b90 00007ff9`0159140d libwireshark!decode_tcp_ports+0x28cd
000000e7`38ef7cf0 00007ff9`00b9ff7e libwireshark!call_dissector_only+0xae
000000e7`38ef7d40 00007ff9`00ba01a1 libwireshark!call_dissector_with_data+0x1b1
000000e7`38ef7da0 00007ff9`00ba32ed libwireshark!dissector_try_uint_new+0x5d
000000e7`38ef7de0 00007ff9`010f1676 libwireshark!ieee802a_add_oui+0x16c46
000000e7`38ef7e50 00007ff9`010ef8d6 libwireshark!ieee802a_add_oui+0x14ea6
000000e7`38ef7f20 00007ff9`00b9ff7e libwireshark!call_dissector_only+0xae
000000e7`38ef7f70 00007ff9`00ba01a1 libwireshark!call_dissector_with_data+0x1b1
000000e7`38ef7fd0 00007ff9`00ba32ed libwireshark!dissector_try_uint_new+0x5d
000000e7`38ef8010 00007ff9`00ba3284 libwireshark!dissector_try_uint+0x24
000000e7`38ef8060 00007ff9`00f374a8 libwireshark!dissect_e212_utf8_imsi+0x34db8
000000e7`38ef82b0 00007ff9`00b9ff7e libwireshark!call_dissector_only+0xae
000000e7`38ef8300 00007ff9`00ba01a1 libwireshark!call_dissector_with_data+0x1b1
000000e7`38ef8360 00007ff9`00b9fef0 libwireshark!call_dissector_only+0x20
000000e7`38ef83a0 00007ff9`00ba0017 libwireshark!call_dissector_with_data+0x27
000000e7`38ef83e0 00007ff9`00f367aa libwireshark!dissect_e212_utf8_imsi+0x340ba
000000e7`38ef84b0 00007ff9`00f36017 libwireshark!dissect_e212_utf8_imsi+0x33927
000000e7`38ef8510 00007ff9`00b9ff7e libwireshark!call_dissector_only+0xae
000000e7`38ef8560 00007ff9`00ba01a1 libwireshark!call_dissector_with_data+0x1b1
000000e7`38ef85c0 00007ff9`00b9fef0 libwireshark!call_dissector_only+0x20
000000e7`38ef8600 00007ff9`00f7922c libwireshark!dissect_e212_utf8_imsi+0x76b3c
000000e7`38ef8a90 00007ff9`00b9ff7e libwireshark!call_dissector_only+0xae
000000e7`38ef8ae0 00007ff9`00ba01a1 libwireshark!call_dissector_with_data+0x1b1
000000e7`38ef8b40 00007ff9`00b9fef0 libwireshark!call_dissector_only+0x20
000000e7`38ef8b80 00007ff9`00ba0017 libwireshark!call_dissector_with_data+0x27
000000e7`38ef8bc0 00007ff9`00ba1333 libwireshark!deregister_depend_dissector+0x9d3
000000e7`38ef8e20 00007ff9`00b96630 libwireshark!epan_dissect_run_with_taps+0x50
000000e7`38ef8e60 00007ff7`6e0547cd Wireshark+0x47cd
000000e7`38ef8ea0 00007ff7`6e0599a6 Wireshark+0x99a6
000000e7`38ef9140 00007ff7`6e056b68 Wireshark+0x6b68
000000e7`38ef9700 00007ff7`6e1783d5 Wireshark+0x1283d5
000000e7`38ef9860 00007ff7`6e074357 Wireshark+0x24357
000000e7`38ef98c0 00007ff7`6e150061 Wireshark+0x100061
000000e7`38ef99a0 00007ff9`0c44e1ac Qt5Widgets!QWidget::event+0x74c
000000e7`38ef9b70 00007ff9`0c428750 Qt5Widgets!QApplicationPrivate::notify_helper+0x140
000000e7`38ef9ba0 00007ff9`0c4272c2 Qt5Widgets!QApplication::notify+0x16b2
000000e7`38efa2a0 00007ff9`0b52a779 Qt5Core!QCoreApplication::notifyInternal2+0xb9
000000e7`38efa320 00007ff9`0c47661b Qt5Widgets!QSizePolicy::QSizePolicy+0x19ab
000000e7`38efa440 00007ff9`0c475997 Qt5Widgets!QSizePolicy::QSizePolicy+0xd27
000000e7`38efa590 00007ff9`0c428750 Qt5Widgets!QApplicationPrivate::notify_helper+0x140
000000e7`38efa5c0 00007ff9`0c42773a Qt5Widgets!QApplication::notify+0x1b2a
000000e7`38efacc0 00007ff9`0b52a779 Qt5Core!QCoreApplication::notifyInternal2+0xb9
000000e7`38efad40 00007ff9`0b9a7851 Qt5Gui!QGuiApplicationPrivate::processDrop+0xb1
000000e7`38efae00 00007ff9`0b9915cb Qt5Gui!QWindowSystemInterface::handleDrop+0x9b
000000e7`38efae80 00007ff9`0ab82a9c qwindows+0x52a9c
000000e7`38efaf40 00007ff9`7137b918 ole32!CPrivDragDrop::PrivDragDrop+0x128
000000e7`38efaf90 00007ff9`6fd72143 rpcrt4!Invoke+0x73
000000e7`38efb030 00007ff9`6fddc1d3 rpcrt4!Ndr64StubWorker+0xb13
000000e7`38efb6d0 00007ff9`6fd1b429 rpcrt4!NdrStubCall3+0xc9
000000e7`38efb730 00007ff9`6f937bc0 combase!CStdStubBuffer_Invoke+0x60
000000e7`38efb770 00007ff9`6f8c6323 combase!ObjectMethodExceptionHandlingAction<<lambda_c9f3956a20c9da92a64affc24fdd69ec> >
```

```
000000e7`38efb7d0 00007ff9`6f8c60ae combase!DefaultStubInvoke+0x1ee
000000e7`38efb930 00007ff9`6f93b166 combase!SyncServerCall::StubInvoke+0x26
000000e7`38efb970 00007ff9`6f8bfdf3 combase!ServerCall::ContextInvoke+0x403
000000e7`38efbcc0 00007ff9`6f8c424d combase!ReentrantSTAInvokeInApartment+0x1ad
000000e7`38efbd40 00007ff9`6f8af1a8 combase!ComInvokeWithLockAndIPID+0xbb8
000000e7`38efc120 00007ff9`6f8e038a combase!ThreadWndProc+0x40a
000000e7`38efc250 00007ff9`70eae858 user32!UserCallWinProcCheckWow+0x2f8
000000e7`38efc3e0 00007ff9`70eae299 user32!DispatchMessageWorker+0x249
000000e7`38efc460 00007ff9`0b574a8e Qt5Core!QEventDispatcherWin32::processEvents+0x5ce
000000e7`38eff5d0 00007ff9`0ab935b9 qwindows!qt_plugin_query_metadata+0x1e79
000000e7`38eff600 00007ff9`0b52652b Qt5Core!QEventLoop::exec+0x1cb
000000e7`38eff680 00007ff9`0b529661 Qt5Core!QCoreApplication::exec+0x141
000000e7`38eff6f0 00007ff7`6e149885 Wireshark+0xf9885
000000e7`38eff8c0 00007ff7`6e350227 Wireshark+0x300227
000000e7`38eff950 00007ff7`6e34f5e6 Wireshark+0x2ff5e6
000000e7`38eff990 00007ff9`6fe46fd4 kernel32!BaseThreadInitThunk+0x14
000000e7`38eff9c0 00007ff9`714dcec1 ntdll!RtlUserThreadStart+0x21
```

It's probably caused by the mistake process to the `tag_offset` , in the `epan/dissectors/packet-fbzero.c` , `dissect_fb_zero_tag` function.

(the `epan/dissectors/packet-fbzero.c` can be found from https://github.com/wireshark/wireshark/blob/98bcda594ecbe3de891bf78cfb262986d1e2b8d6/epan/dissectors/packet-fbzero.c)

The second parameter of `tvb_reported_length_remaining` callsite should be the `offset_end` . In most cases, the tag offset calculated by the `tag_offset_start + tag_offset` is correct.
Nevertheless, when it comes the line 258 in the `packet-fbzero.c` , in that switch case, the `tag` has no implemented case and the `tag_offset` is added by `tag_len` directly. However, in the first time, the `tag_len = offset_end - tag_offset` , `tag_offset = 0` , which means the value of `tag_len` is calculated from the `0` rather than the `tag_offset_start` . The same case happen right after the `switch tag` , in line 268, the `tag_offset` is assigned by the `offset_end` wrongly. The expected way is `tag_offset = offset_end -tag_offset_start`

Due to the wrong calculation of `tag_offset` , upon trigger the defaullt case of `tag` , the `tag offset` will larger than the correct offset. This means the `tvb_reported_length_remaining` will always return `0` because `tag_offset_start + tag_offset` is larger than the packet length. In this way, the `tag_len` will assigned 0 in the line 189. Finally, the total_tag_len is calculated in the wrong way as well and it can controlled by specify several offset in the tcp data frame.

As the `total_tag_len` is controlled in a large number, such as `0xFFFFFF9B` , it will cause Integer Overflow when calculating the return value by `offset + total_tag_len` . In the poc file, I construct eight tag offset and the `total_tag_len` will be `0xFFFFFF9B` and the `offset` is `0x69` , after the unsigned int addition, the result will be `4` , and this offset cause the parent function `fb_zero_unencrypt` process the packet from the offset `4` again, which caused the dead circle. In this circle, the tree node is malloced infinitely and finally use up all free memory in the os.

## Build information

```
Wireshark 3.2.7 (v3.2.7-0-gfb6522d84a3a)

Copyright 1998-2020 Gerald Combs <gerald@wireshark.org> and contributors.
License GPLv2+: GNU GPL version 2 or later <https://www.gnu.org/licenses/gpl-2.0.html>
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiled (64-bit) with Qt 5.12.9, with WinPcap SDK (WpdPack) 4.1.2, with GLib
2.52.3, with zlib 1.2.11, with SMI 0.4.8, with c-ares 1.15.0, with Lua 5.2.4,
with GnuTLS 3.6.3 and PKCS #11 support, with Gcrypt 1.8.3, with MIT Kerberos,
with MaxMind DB resolver, with nghttp2 1.39.2, with brotli, with LZ4, with
Zstandard, with Snappy, with libxml2 2.9.9, with QtMultimedia, with automatic
updates using WinSparkle 0.5.7, with AirPcap, with SpeexDSP (using bundled
resampler), with SBC, with SpanDSP, with bcg729.

Running on 64-bit Windows 10 (2004), build 19041, with Intel(R) Core(TM)
i7-9750H CPU @ 2.60GHz (with SSE4.2), with 16193 MB of physical memory, with
locale Chinese (Simplified)_China.936, with Npcap version 0.9997, based on
libpcap version 1.9.1, with GnuTLS 3.6.3, with Gcrypt 1.8.3, with brotli 1.0.2,
without AirPcap, binary plugins supported (0 loaded).

Built using Microsoft Visual Studio 2019 (VC++ 14.27, build 29111).
```



Related merge requests 4

- FBZERO: Make sure our offset advances. !467
- FBZERO: Make sure our offset advances. !471
- FBZERO: Make sure our offset advances. !472
- FBZERO: Make sure our offset advances. !473

When these merge requests are accepted, this issue will be closed automatically.

## Activity

Gerald Combs mentioned in merge request !467 (merged) 2 years ago

AndersBroman closed via merge request !467 (merged) 2 years ago

Gerald Combs mentioned in commit f97f665d 2 years ago

Alexis La Goutte mentioned in merge request !471 (merged) 2 years ago

Gerald Combs mentioned in commit 334036e5 2 years ago

Alexis La Goutte mentioned in merge request !472 (merged) 2 years ago

Gerald Combs mentioned in commit 2b6648a5 2 years ago

Alexis La Goutte mentioned in merge request !473 (merged) 2 years ago

Alexis La Goutte @alagoutte · 2 years ago — Developer

Thanks, good catch, can you check also packet-gquic.c ? because fbzero is based on gquic....

Gerald Combs mentioned in commit 2bdcb78a 2 years ago

Gerald Combs mentioned in commit e0e40cb7 2 years ago

HAPPY @pcy190 · 2 years ago — Author — Contributor

Fine, I'll check out the packet-gquic.c.
BTW, I wonder whether could I disclosure this issue and apply for a cve id.

Gerald Combs @geraldcombs · 2 years ago — Owner

I normally request CVE IDs before each release, but feel free to do so if you'd like. BTW, how would you prefer to be credited in the Wireshark advisory?

**HAPPY** @pcy190 · 2 years ago — Author — Contributor

Credited name could be `Chaoyuan Peng` ( I'm not sure the way of credit, probably the name is sufficient information)

Please register or sign in to reply

**HAPPY** mentioned in issue #16897 (closed) 2 years ago

**Gerald Combs** made the issue visible to everyone 2 years ago

**Gerald Combs** added crash label 2 years ago

**Gerald Combs** added lib wireshark scoped label 2 years ago

Please register or sign in to reply