Suricata - Bug #1163

HTP Segfault

04/03/2014 08:04 AM - Brad Roether

Status: Closed
Priority: Normal
Assignee: Victor Julien
Category: Target version: 2.0.1rc1
Affected Versions:

Difficult: Effort: Label:

Description

HTP Segfaults - irregular period of time between incidents (have observed cores within as little as 2 minutes or as long as 18+ hours)

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7fff743f7000 (LWP 23342)]
0x00007fff7b7cdf8c in htp_connp_REQ_CONNECT_WAIT_RESPONSE (connp=0x7ffdebef14d0) at htp_request.c:322
322 if (connp->in_tx->response_progress <= HTP_RESPONSE_LINE) {
    (gdb)
    (gdb) bt
    #0 0x00007fff7b7cdf8c in htp_connp_REQ_CONNECT_WAIT_RESPONSE (connp=0x7ffdebef14d0) at htp_request.c:322
    #1 0x00007fff7bce3f9 in htp_connp_req_data (connp=0x7ffdebef14d0, timestamp=<value optimized out>, data=<value optimized out>, len=<value optimized out>) at htp_request.c:851
    pstate=0x7ffdf3043f900, input=0x00007fff7b743c920 "CONNECT tools.google.com:443 HTTP/1.0\r\nHost: tools.google.com\r\nContent-Length: 0\r\nProxy-Connection: Keep-Alive\r\nProxy-Authorization: Negotiate TlRMTVNTUADAAAAAQABAGIAAAAAAYWAAAAAABIAAAAAAEEgAA"
    #2 0x00000000004230b1 in HTPHandleRequestData (f=<value optimized out>, htpp_state=0x7fffa90e4b130, pstate=0x7ffdf3043f900, data=<value optimized out>) at app-layer-htp.c:720
    #3 0x000000000042427da in AppLayerParserParse (alp_tctx=<value optimized out>, f=0x7fff5c98baf0, aproto=1, flags=6 '006', input=<value optimized out>, input_len=<value optimized out>) at app-layer-parser.c:818
    #4 0x0000000000401209 in AppLayerHandleTCPData (tv=0x1390ed00, ra_ctx=0x7fffe0013f5f, p=0x37384600, f=0x7fff75e98baf0, ssn=0x7fffe09deb90, stream=<value optimized out>, data=<value optimized out>, data_len=279, flags=6 '006') at app-layer.c:360
    #5 0x0000000000517875 in StreamTcpReassembleAppLayer (tv=0x1390ed00, ra_ctx=0x7fffe00135f0, ssn=0x7fffe09deb90, stream=0x7fffe09debde0, p=0x37384600) at stream-tcp-reassemble.c:3199
    #6 0x0000000000517d00 in StreamTcpReassembleHandleSegmentUpdateACK (tv=0x1390ed00, ra_ctx=0x7fffe00135f0, ssn=0x7fffe09deb90, stream=0x7fffe09debde0, p=0x37384600) at stream-tcp-reassemble.c:3545
    #7 0x0000000000519e9a in StreamTcpReassembleHandleSegment (tv=0x1390ed00, ra_ctx=0x7fffe00135f0, ssn=0x7fffe09deb90, stream=0x7fffe09deb98, p=0x3738460, pq=<value optimized out>) at stream-tcp-reassemble.c:3573
    #8 0x00000000005146e5 in StreamTcpPacket (tv=0x1390ed00, p=0x43734a0, ssn=0x7fffe0012f00, pg=<value optimized out>) at stream-tcp.c:4363
    #9 0x0000000000515e2c in StreamTcp (tv=0x1390ed00, p=0x43734a0, data=0x7fffe0012f00, pg=0x137ecb20, postpq=<value optimized out>) at stream-tcp.c:4485
    #10 0x000000000052a4d0 in TmThreadsSlotVarRun (tv=0x1390ed00, slot=<value optimized out>) at tm-threads.c:559
    #11 0x000000000050bdef in TmThreadsSlotProcessPkt (tv=0x1390ed00, data=<value optimized out>, slot=<value optimized out>) at source-pfring.c:361
    #12 ReceivePfringLoop (tv=0x1390ed00, data=<value optimized out>, slot=<value optimized out>) at source-pfring.c:361
    #13 0x000000000052e1e in TmThreadsSlotPktAcqLoop (td=0x1390ed00) at tm-threads.c:703
    #14 0x00007fff51be9d1 in start_thread (arg=0x7fffe0743f700) at pthread_create.c:301

07/16/2022
This is Suricata version 2.0 RELEASE
Features: PCAP_SET_BUFF LIBPCAP_VERSION_MAJOR=1 PF_RING HAVE_PACKET_FANOUT LIBCAP_LIBNET1.1 HAVE_HTP_URL_NORMALIZE_HOOK PCRE_JIT HAVE_LIBNSS HAVE_LIBJUJIT HAVE_LIBJANSSON PROFILING
SIMD support: SSE_4_2 SSE_4_1 SSE_3
Atomic intrinsics: 1 2 4 8 16 byte(s)
64-bits, Little-endian architecture
GCC version 4.4.7 20120313 (Red Hat 4.4.7-4), C version 199901
compiled with -fstack-protector
compiled with _FORTIFY_SOURCE=2
L1 cache line size (CLS)=64
compiled with LibHTP v0.5.10, linked against LibHTP v0.5.10
Suricata Configuration:
   AF_PACKET support: no
   PF_RING support: yes
   NFQueue support: no
   IPFW support: no
   DAG enabled: no
   Napatech enabled: no
   Unix socket enabled: yes
   Detection enabled: yes
   libnss support: yes
   libnspr support: yes
   libjansson support: yes
   Prelude support: no
   PCRE jit: yes
   libluajit: yes
   libgeoip: yes
   Non-bundled htp: no
   Old barnyard2 support: no
   CUDA enabled: no

Suricatasc install: yes

Unit tests enabled: no
Debug output enabled: no
Debug validation enabled: no
Profiling enabled: yes
Profiling locks enabled: no
Coccinelle / spatch: no

Generic build parameters:
   Installation prefix (--prefix): [ REDACTED ]
   Configuration directory (--sysconfdir): [ REDACTED ]
   Log directory (--localstatedir): [ REDACTED ]

   Host: x86_64-unknown-linux-gnu
   GCC binary: gcc
   GCC Protect enabled: yes
   GCC march native enabled: yes
   GCC Profile enabled: no

Other configuration items of interest:
- Using PF_RING build 7180 w/ DNA driver on Intel 82599 NIC
- Suricata is using runmode: workers and cluster_flow
- Included IRQ affinity script is being used
- ixgbe parameters: MQ=1,1 RSS=16,16 FdirPballoc=3,3 num_rx_slots=32768 mtu=1500
- pf_ring parameters: transparent_mode=2 quick_mode=1 enable_frag_coherence=1 min_num_slots=65536 enable_tx_capture=0 enable_ip_defrag=0

History
#1 - 04/03/2014 08:17 AM - Brad Roether
Core was generated by `/usr/src/pf_ring/bin/suricata --pfring -Dv /usr/src/pf_ring/etc/suricata/suricata'.
Program terminated with signal 11, Segmentation fault.

#0 0x00007fa7f7caef8c in htp_connp_REQ_CONNECT_WAIT_RESPONSE (connp=0x7fa7e5858830) at http_request.c:322
#1 0x00007fa7f7caaf3f9 in htp_connp_req_data (connp=0x7fa7e5858830, timestamp=<value optimized out>, data=<value optimized out>, len=<value optimized out>) at http_request.c:851

Program terminated with signal 11, Segmentation fault.

#0 0x00007fa7f7caef8c in htp_connp_REQ_CONNECT_WAIT_RESPONSE (connp=0x7fa7e5858830) at http_request.c:322
322 if (connp->in_tx->response_progress <= HTP_RESPONSE_LINE) {
    len<value optimized out>) at http_request.c:322

322 if (connp->in_tx->response_progress <= HTP_RESPONSE_LINE) {
    len<value optimized out>) at http_request.c:322

#1 0x00000000004230b1 in HTPHandleRequestData (f=<value optimized out>, htp_state=0x7fa6c875c610, pstate=0x7fa7e5858a40, input=0x7fa7f7bd7920 "GET http://clients5.google.com/complete/search?hl=en-us&q=g&client=ie8&inputencoding=UTF-8&outputencoding=UTF-8 HTTP/1.1
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE)...

....
}, data_len=690, flags=6 ('006') at app-layer.c:360

#5 0x0000000000517875 in StreamTcpReassembleAppLayer (tv=0x7fa702d9450, f=0x7fa702d9450, pstate=0x7fa702d9450a0, stream=0x7fa702d9450f0, p=0xebe2c8f0)

#6 0x0000000000517d00 in StreamTcpReassembleHandleSegmentUpdateACK (tv=0x13d9ff0, ra_ctx=0x7fa702d9450, stream=0x7fa702d9450f0, p=0xebe2c8f0)

#7 0x0000000000519e9a in StreamTcpReassembleHandleSegment (tv=0x13d9ff0, ra_ctx=0x7fa702d9450, stream=0x7fa702d9450f0, p=0xebe2c8f0, pq=<value optimized out>) at stream-tcp-reassemble.c:3573

#8 0x00000000005146e5 in StreamTcpPacket (tv=0x13d9ff0, p=0xebe9f310, stt=0x7fa70b158af0, pq=0x161274b0) at stream-tcp.c:4363

#9 0x0000000000514665 in StreamTcpPacket (tv=0x13d9ff0, p=0xebe9f310, data=0x7fa70b101f0, pq=0x161274b0, postpq=<value optimized out>) at stream-tcp.c:4485

#10 0x000000000052a4d0 in TmThreadsSlotVarRun (tv=0x13d9ff0, data=<value optimized out>, slot=<value optimized out>) at tm-threads.c:559

#11 0x000000000050bdef in TmThreadsSlotProcessPkt (tv=0x13d9ff0, data=<value optimized out>, slot=<value optimized out>) at tm-threads.h:142

#12 0x0000000000517875 in StreamTcpReassembleAppLayer (tv=0x7fa702d9450, f=0x7fa702d9450, pstate=0x7fa702d9450a0, stream=0x7fa702d9450f0, p=0xebe2c8f0)

#13 0x0000000000517d00 in Stream TcpReassembleHandleSegmentUpdateACK (tv=0x13d9ff0, ra_ctx=0x7fa702d9450, stream=0x7fa702d9450f0, p=0xebe2c8f0)

#14 0x0000000000517d00 in StreamTcpReassembleHandleSegmentUpdateACK (tv=0x13d9ff0, ra_ctx=0x7fa702d9450, stream=0x7fa702d9450f0, p=0xebe2c8f0)

#15 0x0000000000517875 in StreamTcpReassembleAppLayer (tv=0x7fa702d9450, f=0x7fa702d9450, pstate=0x7fa702d9450a0, stream=0x7fa702d9450f0, p=0xebe2c8f0)

07/16/2022
(gdb) print *((HtpTxUserData *)connp->user_data)
$4 = {request_body_init = 48 '0', response_body_init = 136 '\210', request_body = {first = 0x7fa70b158af0, last = 0x7fa75d0ca8f0, content_len = 1,
  content_len_so_far = 0, body_parsed = 0, body_inspected = 0}, response_body = {first = 0x7e9ba0, last = 0x7e9ca0, content_len = 373, content_len_so_far = 323,
  body_parsed = 323, body_inspected = 0}, request_uri_normalized = 0x72646c6968633d71,
  request_headers_raw = 0x68413325666f6e65 <Address 0x68413325666f6e65 out of bounds>,
  response_headers_raw = 0x3225413325707477 <Address 0x3225413325707477 out of bounds>,
  request_headers_raw_len = 1177691462, response_headers_raw_len = 779581303,
  decoder_events = 0x6f6f632e6f706377, boundary = 0x6874616577463225 <Address 0x6874616577463225 out of bounds>,
  boundary_len = 101 'e', tsflags = 114 'r',
  tcflags = 37 '%', operation = 13382, request_body_type = 48 '0', response_body_type = 45 '-'}

#3 - 04/03/2014 09:02 AM - Anoop Saldanha
Just wondering if it is the below situation -

The request is still waiting on the response. The response comes in, the response progress moves forward.

Meanwhile suricata doesn't consider the inspect_ids for both directions while culling the transaction(I recollect we checking both directions, but the code seems to have changed to consider the direction flags). The response progress might have moved on to the next one, and we end up culling the transaction, which libhtp is still parsing.

I think this should be reproducible even with non CONNECT requests.

#4 - 04/03/2014 10:03 AM - Ivan Ristic

Anoop Saldanha wrote:

Just wondering if it is the below situation -

The request is still waiting on the response. The response comes in, the response progress moves forward.

Meanwhile suricata doesn't consider the inspect_ids for both directions while culling the transaction(I recollect we checking both directions, but the code seems to have changed to consider the direction flags). The response progress might have moved on to the next one, and we end up culling the transaction, which libhtp is still parsing.

I think this should be reproducible even with non CONNECT requests.

Could this be a race condition where htp_connp_destroy_all() is called and another thread attempts to send more data on the same connection?

I see in the Suricata code that you destroy transactions in one of two ways: htp_tx_destroy() and htp_connp_destroy_all(). But the former will not allow you to destroy an incomplete transaction, which leaves the other path. From the crashdump, we know that both in_tx and out_tx are NULL, but both connection states are inconsistent with that.

Here's what might have happened:

1. CONNECT request comes in (req); LibHTP parses it and stops to see the response. The state is REQ_CONNECT_WAIT_RESPONSE.
2. CONNECT request has been rejected (headers 299 bytes, body 666); LibHTP parses it, sees that the other side is waiting for the response, and returns. The state is RES_FINALIZE, but notice that the previous state on that connection is also RES_FINALIZE. This means that the state was already invoked at least once, but couldn't complete.

Speculation:

3. htp_connp_destroy_all() is invoked. While it is executing:
4. The next request comes in, but in_tx and out_tx are already NULL. Some other values are not yet NULL, which is why we don't see a segfault earlier.

The reason I suspect this is that connp->conn is not (yet) NULL.

#5 - 04/03/2014 10:06 AM - Ivan Ristic
Brad, could we please also get a dump of connp->conn? Thanks.

#6 - 04/03/2014 10:38 AM - Brad Roether
Per Victor's request in IRC:

(gdb) thread apply all bt
Thread 21 (Thread 0x7fa7b751f700 (LWP 16237)):
#0 0x00007fa7f4dd4343 in __poll (fds=<value optimized out>, nfds=<value optimized out>, timeout=<value optimized out>) at ../sysdeps/unix/sysv/linux/poll.c:87
#1 0x00007fa7f71ac713 in pfring_mod_poll () from /usr/src/pf_ring/lib/libpcap.so.1

07/16/2022
The user data dump seems to contain only randomness, that is odd.

The threads don't show that another thread is doing something with that HTP state at the exact same time.

Ivan Ristic wrote:
Brad, could we please also get a dump of connp->conn? Thanks.

(gdb) print connp->conn
$1 = (htp_conn_t *) 0x7fa70b158af0

#9 - 04/03/2014 10:48 AM - Brad Roether
(gdb) print *connp->conn
$2 = {client_addr = 0x0, client_port = 2785, server_addr = 0x0, server_port = 8080, transactions = 0x7fa7e5858dc0, messages = 0x7fa7e5cbed90, flags = 0 '000',
        open_timestamp = {tv_sec = 1396488355, tv_usec = 984693}, close_timestamp = {tv_sec = 0, tv_usec = 0}, in_data_counter = 1020, out_data_counter = 965}

#10 - 04/03/2014 12:10 PM - Ivan Ristic
Victor Julien wrote:

The user data dump seems to contain only randomness, that is odd.

The threads don't show that another thread is doing something with that HTP state at the exact same time.

In that case, I think Anoop's theory is the next likely explanation. I've just discovered that a partial transaction cleanup is invoked on CONNECT requests, causing the request to be marked as complete. After the response is seen, invoking htp_tx_destroy() will clear both in_tx and out_tx, leading to the segfault when the next request arrives.

I see two improvements to LibHTP here (neither of which would fix the problem, but it would at least make it fail gracefully): (1) do not mark request as complete until it's actually complete and (2) check in_tx and out_tx when new data arrives.

#11 - 04/04/2014 02:23 AM - Victor Julien
- Status changed from New to Assigned
- Assignee set to Victor Julien
- Target version set to 2.0.1rc1

We mark a tx for removal when it's progress is complete. So we check to see if htp_tx_t::progress is HTP_REQUEST_COMPLETE (for our request inspection) or HTP_RESPONSE_COMPLETE (for our response inspection). If the progress is complete, we update the active_tx to move beyond the current tx. When both sides have moved beyond a tx, we remove it by ultimately calling htp_tx_destroy(). So our handling depends on correct progress tracking in htp.

I don't see any other paths leading to us calling htp_tx_destroy(), but I'll give it some more thought as well.

#12 - 04/04/2014 02:23 AM - Victor Julien
- Description updated

#13 - 04/04/2014 03:29 AM - Ivan Ristic
Victor Julien wrote:

We mark a tx for removal when it's progress is complete. So we check to see if htp_tx_t::progress is HTP_REQUEST_COMPLETE (for our request inspection) or HTP_RESPONSE_COMPLETE (for our response inspection). If the progress is complete, we update the active_tx to move beyond the current tx. When both sides have moved beyond a tx, we remove it by ultimately calling htp_tx_destroy(). So our handling depends on correct progress tracking in htp.

That makes sense. I will fix it soon (hopefully today) and release 0.5.11.

#14 - 04/05/2014 10:29 AM - Ivan Ristic
FYI: LibHTP v0.5.11 has been released: https://github.com/ironbee/libhtp/releases/tag/0.5.11

#15 - 04/10/2014 06:06 AM - Victor Julien
Brad, have you been able to test the new libhtp?

#16 - 04/10/2014 08:56 AM - Brad Roether
Yes, this issue does appear to be fixed - thank you!

#17 - 04/10/2014 08:57 AM - Victor Julien
- Status changed from Assigned to Closed
Great, thanks Brad and thanks Ivan!