## Introduction

As described in <https://redmine.openinfosecfoundation.org/issues/6782> we have a number of customers experiencing Suricata crashes. The backtrace from these crashes point towards HTTP traffic. We have been supplied with a customer *pcap* as a reproducer and have conducted an investigation into the reason for this crash. Suricata was built and installed with AddressSanitizer and run under gdb and valgrind during this investigation.

## Findings

The following output was produced during the segfault:

==83833==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x627000145100 at pc 0x7ffff786e9ef bp 0x7fffe29fe6e0 sp 0x7fffe29fdea0

WRITE of size 2920 at 0x627000145100 thread T1 (W#01)

 #0 0x7ffff786e9ee in \_\_interceptor\_memcpy (/lib64/libasan.so.8+0x6e9ee) (BuildId: 2b657470ea196ba4342e3bd8a3cc138b1e200599)

 #1 0x7c1295 in memcpy /usr/include/bits/string\_fortified.h:29

 #2 0x7c1295 in StreamingBufferAppend /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/util-streaming-buffer.c:1087

 #3 0x7fc694 in HtpBodyAppendChunk /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/app-layer-htp-body.c:71

 #4 0x579cb4 in HTPCallbackResponseBodyData /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/app-layer-htp.c:2026

 #5 0x7ffff7f67e43 in htp\_hook\_run\_all /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/libhtp/htp/htp\_hooks.c:127

 #6 0x7ffff7f8223d in htp\_tx\_res\_process\_body\_data\_ex /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/libhtp/htp/htp\_transaction.c:1005

 #7 0x7ffff7f7611d in htp\_connp\_RES\_BODY\_IDENTITY\_CL\_KNOWN /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/libhtp/htp/htp\_response.c:490

 #8 0x7ffff7f7bcb4 in htp\_connp\_res\_data (/lib64/libhtp.so.2+0x2ecb4) (BuildId: 01dcb6a8931edaab9119620e495b29935cfceab9)

 #9 0x5803d9 in HTPHandleResponseData /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/app-layer-htp.c:970

 #10 0x58dad4 in AppLayerParserParse /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/app-layer-parser.c:1403

 #11 0x56514c in AppLayerHandleTCPData /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/app-layer.c:787

 #12 0x772c67 in ReassembleUpdateAppLayer /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp-reassemble.c:1328

 #13 0x775ed9 in StreamTcpReassembleAppLayer /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp-reassemble.c:1391

 #14 0x776008 in StreamTcpReassembleHandleSegmentUpdateACK /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp-reassemble.c:1949

 #15 0x777602 in StreamTcpReassembleHandleSegment /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp-reassemble.c:1997

 #16 0x752450 in HandleEstablishedPacketToServer /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp.c:2666

 #17 0x75545b in StreamTcpPacketStateEstablished /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp.c:3209

 #18 0x766bf0 in StreamTcpStateDispatch /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp.c:5236

 #19 0x767670 in StreamTcpPacket /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp.c:5433

 #20 0x767c51 in StreamTcp /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/stream-tcp.c:5745

 #21 0x6cc42c in FlowWorkerStreamTCPUpdate /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/flow-worker.c:391

 #22 0x6cda1e in FlowWorker /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/flow-worker.c:607

 #23 0x52c653 in TmThreadsSlotVarRun /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/tm-threads.c:135

 #24 0x73ba89 in TmThreadsSlotProcessPkt /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/tm-threads.h:200

 #25 0x73be4d in PcapFileCallbackLoop /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/source-pcap-file-helper.c:108

 #26 0x7ffff74b15ae in pcap\_offline\_read (/lib64/libpcap.so.1+0x2d5ae) (BuildId: b2c27ef72665c5895f38a2abd1ea7e63b2962439)

 #27 0x73c7a7 in PcapFileDispatch /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/source-pcap-file-helper.c:153

 #28 0x737e0e in ReceivePcapFileLoop /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/source-pcap-file.c:180

 #29 0x52f73f in TmThreadsSlotPktAcqLoop /usr/src/debug/suricata-rapid7-7.0.3-1.fc39.x86\_64/src/tm-threads.c:318

 #30 0x7ffff70ac896 in start\_thread (/lib64/libc.so.6+0x8e896) (BuildId: 0d710e9d9dc10c500b8119c85da75004183618e2)

The following conclusions were made from investigating further:

* Comparing http.memcap between our customer suricata.yaml and the installed suricata.yaml showed we do not use the default (unlimited) http.memcap, rather cap this to 512mb.
* HTPConfigure() defines:
	+ htp\_sbcfg.Calloc = HTPCalloc()
	+ htp\_sbcfg.Realloc = HTPRealloc()
	+ htp\_sbcfg.Free = HTPFree()
* The above functions return NULL if there is no memory left or malloc/calloc fails
* GrowRegionToSize() returns sc\_errno if REALLOC() (a macro that uses the above functions) returns NULL
* InitBuffer() returns sc\_errno if CALLOC() (a macro that uses the above functions) returns NULL
* The following three functions do not set sc\_errno upon error return:
	+ HTPCalloc()
	+ HTPRealloc()
	+ HTPFree()

Given the above findings, I was able to test and confirm there is a scenario where callers to the above functions return ‘0’ where coincidentally SC\_OK = 0; The two functions of concern here are:

* GrowRegionToSize()
* InitBuffer()

And therefore the callers to these receive SC\_OK and continue, resulting in the segmentation fault shown above. The scenario is simple:

1. Set a low http.memcap (1mb for my test, but also failed at 512mb)
2. Pass HTTP traffic that requires reassembly to Suricata until there is no memory left
3. Pass more HTTP traffic that requires reassembly to Suricata

Once a new packet/message arrives it will fail to allocate memory resulting in a return NULL; from the allocator(s). Given the above findings, a return NULL; from these functions results in the caller (in this case StreamingBufferAppend()) being provided with a ‘0’ or SC\_OK.

To test this theory, I updated the above allocators to set sc\_errno = SC\_ENOMEM; upon return NULL; (see the diff below). With these changes Suricata was able to process the file without segmentation faults with http.memcap = 1mb.

diff --git a/src/app-layer-htp-mem.c b/src/app-layer-htp-mem.c

index bd9b79f67..d61b42cb2 100644

--- a/src/app-layer-htp-mem.c

+++ b/src/app-layer-htp-mem.c

@@ -136,13 +136,17 @@ void \*HTPMalloc(size\_t size)

 {

 void \*ptr = NULL;

- if (HTPCheckMemcap((uint32\_t)size) == 0)

- return NULL;

+ if (HTPCheckMemcap((uint32\_t)size) == 0) {

+ sc\_errno = SC\_ENOMEM;

+ return NULL;

+ }

 ptr = SCMalloc(size);

- if (unlikely(ptr == NULL))

+ if (unlikely(ptr == NULL)) {

+ sc\_errno = SC\_ENOMEM;

 return NULL;

+ }

 HTPIncrMemuse((uint64\_t)size);

@@ -153,13 +157,17 @@ void \*HTPCalloc(size\_t n, size\_t size)

 {

 void \*ptr = NULL;

- if (HTPCheckMemcap((uint32\_t)(n \* size)) == 0)

+ if (HTPCheckMemcap((uint32\_t)(n \* size)) == 0) {

+ sc\_errno = SC\_ENOMEM;

 return NULL;

+ }

 ptr = SCCalloc(n, size);

- if (unlikely(ptr == NULL))

+ if (unlikely(ptr == NULL)) {

+ sc\_errno = SC\_ENOMEM;

 return NULL;

+ }

 HTPIncrMemuse((uint64\_t)(n \* size));

@@ -169,13 +177,17 @@ void \*HTPCalloc(size\_t n, size\_t size)

 void \*HTPRealloc(void \*ptr, size\_t orig\_size, size\_t size)

 {

 if (size > orig\_size) {

- if (HTPCheckMemcap((uint32\_t)(size - orig\_size)) == 0)

+ if (HTPCheckMemcap((uint32\_t)(size - orig\_size)) == 0) {

+ sc\_errno = SC\_ENOMEM;

 return NULL;

+ }

 }

 void \*rptr = SCRealloc(ptr, size);

- if (rptr == NULL)

+ if (rptr == NULL) {

+ sc\_errno = SC\_ENOMEM;

 return NULL;

+ }



## Conclusion

These findings reduce to a failure of setting sc\_errno upon error return by the *HTPAllocators* (above). This fix may not be a complete fix for Suricata to correctly operate given additional functionality (functionality I’m unfamiliar with) within the system. These changes were completed to prove the above theory only. However, by reading ReallocFunc() and CallocFunc() in util-streaming-buffer.c - the default allocators if none are defined - they set sc\_errno = SC\_ENOMEM; upon error, so it’s possible this fix may be acceptable?

Options moving forward are:

1. We set http.memcap to default (unlimited)
2. A fix is produced for the above issue