Suricata - Bug #4478
freebsd: lockups due to mutex handling issues
05/06/2021 02:41 PM - Bill Meeks

Status: Closed
Priority: Normal
Assignee: Victor Julien
Category: Target version: 7.0rc1
Affected Versions: 6.0.0, 6.0.1, 6.0.2, 6.0.3, git master
Effort: Difficulty:
Label: Needs backport to 6.0

Description

Some change between Suricata 5.x and Suricata 6.x in the netmap operation code seems to have introduced an unintended bug. On FreeBSD-12 I am seeing a netmap-enabled interface in Suricata 6.0.2 suddenly stop passing traffic. Nothing is logged to either the console nor the system log when this happens, but the interface will not pass any traffic until Suricata is stopped and the netmap-enabled interface is then cycled back to normal operation. The particular mode of operation I am using is a NIC-to-Host Stack arrangement for IPS mode operation.

I am testing in a VMware virtual machine using the e1000 virtual NIC. I am actually testing on the latest editions of pfSense (which is based on FreeBSD-12.2/STABLE), but I strongly suspect the problem can be reproduced on any FreeBSD-12 installation. The exact same Suricata configuration on the exact same virtual machine using Suricata 5.0.6 works just fine, so the problem appears to definitely be in the Suricata 6.0 code. I investigated using `diff` to compare changes in the relevant source files between Suricata 5.0.6 and Suricata 6.0.2. The only change I see that could possibly be related is the change to the thread packet queue logic. I don't fully understand what that change was about, but it really appears to be the only substantial change in the Suricata code for netmap between versions 5.x and 6.x. What appears to happen is that Suricata stops transmitting packets at some point when in IPS mode.

I noticed this bug while working on migrating the Suricata package for the pfSense firewall distro (which I maintain) from the 5.x binary branch to the 6.x binary branch.

Related issues:
Copied to Bug #4629: freebsd: lockups due to mutex handling issues

History

#1 - 06/22/2021 06:02 AM - Victor Julien
What happens if you compile Suricata with --enable-debug and set the following env vars: SC_LOG_LEVEL=Debug and SC_LOG_OP_FILTER=netmap?

#2 - 06/22/2021 01:39 PM - Victor Julien
- Status changed from New to Assigned
- Assignee set to Jeff Lucovsky
- Priority changed from High to Normal
- Target version set to 6.0.4

#3 - 06/22/2021 10:48 PM - Bill Meeks
I have some additional information to report. Don't know how useful it might be, though.

After some period of time (I suspect the length of time may be dependent on the amount of traffic through the interface), the interface stops passing traffic. Looking at `top` in FreeBSD after the interface freezes, I see the Suricata process is hung in a kernel lock. During that spin lock, the Suricata application is completely unresponsive. You can't kill it with CTRL+C. You must open another shell session and actually use `kill` on the PID. While grabbing the output of `procstat` on the spinlock PID using an additional SSH shell session, suddenly the Suricata process (originally run from the console) began spewing netmap errors to the console. It was still completely unresponsive, though, to keyboard input. The netmap error message was:

```plaintext
netmap_transmit em0 full hwcur 821 hwtail 821 qlen 1023
```

The error was scrolling continuously on the console. This error message is from netmap itself in the kernel, and generally means the rings are not being emptied and all the buffers are full. Thinking about it, this would not be unexpected if the Suricata process that is supposed to be emptying the rings is itself stuck in a kernel lock. So this netmap message is likely just a consequence of the former kernel lock. And the fact the netmap rings were not being processed explains the stoppage of network traffic on the interface.
Stats are updated, but capture.kernel_packets does not rise after few minutes of running.

Here is the output of `procstat` on the PID during the kernel lock spin.

```
<table>
<thead>
<tr>
<th>PID</th>
<th>TID</th>
<th>COMM</th>
<th>TDNAME</th>
<th>KSTACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>15994</td>
<td>100213</td>
<td>suricata</td>
<td>SuriCata-Main</td>
<td>mi_switch sleepq_catch_signals sleepq_wait_slot sleep umt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>xq_sleep do_lock_umutex __umtx_op_wait_umutex sys_umtx_op amd64_syscall fast_syscall_common</td>
</tr>
<tr>
<td>15994</td>
<td>100616</td>
<td>suricata</td>
<td>RX#01-em0</td>
<td>mi_switch sleepq_catch_signals sleepq_timedwait_slot sleep umt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p kern_clock_nanosleep sys_nanosleep amd64_syscall fast_syscall_common</td>
</tr>
<tr>
<td>15994</td>
<td>100623</td>
<td>suricata</td>
<td>RX#01-em0^</td>
<td>mi_switch sleepq_catch_signals sleepq_timedwait_slot sleep umt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p kern_clock_nanosleep sys_nanosleep amd64_syscall fast_syscall_common</td>
</tr>
<tr>
<td>15994</td>
<td>100624</td>
<td>suricata</td>
<td>W#01</td>
<td>mi_switch sleepq_catch_signals sleepq_wait_slot sleep umt</td>
</tr>
<tr>
<td>15994</td>
<td>100625</td>
<td>suricata</td>
<td>W#02</td>
<td>mi_switch sleepq_catch_signals sleepq_wait_slot sleep umt</td>
</tr>
<tr>
<td>15994</td>
<td>100626</td>
<td>suricata</td>
<td>W#03</td>
<td>mi_switch sleepq_catch_signals sleepq_wait_slot sleep umt</td>
</tr>
<tr>
<td>15994</td>
<td>100627</td>
<td>suricata</td>
<td>W#04</td>
<td>mi_switch sleepq_catch_signals sleepq_wait_slot sleep umt</td>
</tr>
<tr>
<td>15994</td>
<td>100628</td>
<td>suricata</td>
<td>P#01</td>
<td>mi_switch sleepq_catch_signals sleepq_timedwait_slot sleep umt</td>
</tr>
<tr>
<td>15994</td>
<td>100629</td>
<td>suricata</td>
<td>FR#01</td>
<td>mi_switch sleepq_catch_signals sleepq_timedwait_slot sleep umt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p kern_clock_nanosleep sys_nanosleep amd64_syscall fast_syscall_common</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p kern_clock_nanosleep sys_nanosleep amd64_syscall fast_syscall_common</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p kern_clock_nanosleep sys_nanosleep amd64_syscall fast_syscall_common</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p kern_clock_nanosleep sys_nanosleep amd64_syscall fast_syscall_common</td>
</tr>
</tbody>
</table>
```

The Suricata process remained in the kernel lock, and completely unresponsive to user input, until I killed the process from the other SSH shell session.

This was a rather simple IPS setup using em0,em0^ as the netmap endpoints. The test was done on a VMWare Workstation virtual machine using e1000 emulated NICS under pfSense-2.6.0.DEVEL (which is based on FreeBSD-12.2-STABLE). Debugging was enabled, but absolutely nothing out of the ordinary was printed with the provided log filter in place (SC_LOG_OP_FILTER = netmap).

---

**#4 - 06/24/2021 11:05 PM - Bill Meeks**

Just to eliminate the possibility of pfSense being part of the issue, I installed Suricata from Ports on a fresh FreeBSD-12.2/STABLE virtual machine. I was able to reproduce the "hang" issue using both "out-of-the-box" Suricata-6.0.2_2 from FreeBSD Ports, and I also modified Suricata to use the new v14 Netmap API in FreeBSD, but had the exact same hang condition occur. It takes some amount of time for the hang to occur. It will usually happen within 5 to 6 minutes on a low-traffic machine. I had Suricata in IPS mode using netmap with em0 and em0^ as the two interfaces. All rules were just "alert" for testing.

The thread named "RX#01-em0" shows a status of "umtxn" when running `procstat -t <PID>` (where `<PID>` is the Suricata process ID).

This problem does not occur with the Suricata 5.0.6 or 5.0.5 binary.

---

**#5 - 07/01/2021 08:52 AM - Martin Rehak**

- *File out.txt added*

Hi, I have the same issue on FreeBSD 13.0-RELEASE-p1 with suricata-6.0.2_3 in IDS pcap mode. Netmap is not used.

Stats are updated, but capture.kernel_packets does not rise after few minutes of running.

```
<table>
<thead>
<tr>
<th>Counter</th>
<th>TM Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture.kernel_packets</td>
<td>Total</td>
<td>17165</td>
</tr>
<tr>
<td>decoder.pkts</td>
<td>Total</td>
<td>17165</td>
</tr>
<tr>
<td>decoder.bytes</td>
<td>Total</td>
<td>4987361</td>
</tr>
<tr>
<td>decoder.ipv4</td>
<td>Total</td>
<td>14943</td>
</tr>
<tr>
<td>decoder.ipv6</td>
<td>Total</td>
<td>1365</td>
</tr>
<tr>
<td>decoder.ethernet</td>
<td>Total</td>
<td>17165</td>
</tr>
<tr>
<td>decoder.tcp</td>
<td>Total</td>
<td>15920</td>
</tr>
<tr>
<td>decoder.udp</td>
<td>Total</td>
<td>875</td>
</tr>
<tr>
<td>decoder.icmpv4</td>
<td>Total</td>
<td>93</td>
</tr>
<tr>
<td>decoder.icmpv6</td>
<td>Total</td>
<td>290</td>
</tr>
<tr>
<td>decoder.avg_pkt_size</td>
<td>Total</td>
<td>1514</td>
</tr>
<tr>
<td>decoder.max_pkt_size</td>
<td>Total</td>
<td>48</td>
</tr>
<tr>
<td>flow.tcp</td>
<td>Total</td>
<td>274</td>
</tr>
<tr>
<td>flow.udp</td>
<td>Total</td>
<td>2</td>
</tr>
<tr>
<td>flow.icmpv4</td>
<td>Total</td>
<td>23</td>
</tr>
<tr>
<td>flow.icmpv6</td>
<td>Total</td>
<td>100</td>
</tr>
<tr>
<td>flow.wrk.spare_sync_avg</td>
<td>Total</td>
<td>12</td>
</tr>
<tr>
<td>flow.wrk.spare_sync</td>
<td>Total</td>
<td>6</td>
</tr>
<tr>
<td>flow.wrk.flows_evicted_needs_work</td>
<td>Total</td>
<td>6</td>
</tr>
<tr>
<td>flow.wrk.flows_evicted_pkt_inject</td>
<td>Total</td>
<td>6</td>
</tr>
</tbody>
</table>
```

**07/17/2022**
<table>
<thead>
<tr>
<th>Metric</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow.wrk.flows_evicted</td>
<td>23</td>
</tr>
<tr>
<td>flow.wrk.flows_injected</td>
<td>6</td>
</tr>
<tr>
<td>tcp.sessions</td>
<td>40</td>
</tr>
<tr>
<td>tcp.syn</td>
<td>40</td>
</tr>
<tr>
<td>tcp.synack</td>
<td>40</td>
</tr>
<tr>
<td>tcp.rst</td>
<td>13</td>
</tr>
<tr>
<td>tcp.overlap</td>
<td>1</td>
</tr>
<tr>
<td>detect.alert</td>
<td>1</td>
</tr>
<tr>
<td>app_layer.flow.http</td>
<td>9</td>
</tr>
<tr>
<td>app_layer.tx.http</td>
<td>9</td>
</tr>
<tr>
<td>app_layer.flow.tls</td>
<td>31</td>
</tr>
<tr>
<td>app_layer.flow.ntp</td>
<td>6</td>
</tr>
<tr>
<td>app_layer.tx.ntp</td>
<td>11</td>
</tr>
<tr>
<td>app_layer.flow.dns_udp</td>
<td>182</td>
</tr>
<tr>
<td>app_layer.tx.dns_udp</td>
<td>364</td>
</tr>
<tr>
<td>app_layer.flow.failed_udp</td>
<td>86</td>
</tr>
<tr>
<td>flow.mgr.full_hash_pass</td>
<td>4</td>
</tr>
<tr>
<td>flow.spare</td>
<td>10292</td>
</tr>
<tr>
<td>flow.mgr.rows_maxlen</td>
<td>2</td>
</tr>
<tr>
<td>flow.mgr.flows_checked</td>
<td>553</td>
</tr>
<tr>
<td>flow.mgr.flows_notimeout</td>
<td>230</td>
</tr>
<tr>
<td>flow.mgr.flows_timeout</td>
<td>323</td>
</tr>
<tr>
<td>flow.mgr.flows_evicted</td>
<td>323</td>
</tr>
<tr>
<td>flow.mgr.flows_evicted_needs_work</td>
<td>30</td>
</tr>
<tr>
<td>tcp.memuse</td>
<td>6881280</td>
</tr>
<tr>
<td>tcp.reassembly_memuse</td>
<td>1339392</td>
</tr>
<tr>
<td>flow.memuse</td>
<td>7419904</td>
</tr>
</tbody>
</table>

```
# procstat arguments 54318
PID COMM     ARGS
54318 suricata suricata --pcap=igb0 -vvv
```

```
# procstat thread 54318
PID TID COMM    TDNAME          CPU PRI STATE         WCHAN
54318 104027 suricata Suricata-Main -1 120 sleep     nanslp
54318 117465 suricata RX#01-igb0 -1 120 sleep     nanslp
uwait
54318 117466 suricata W#01        -1 120 sleep     umtxn
54318 117467 suricata W#02        -1 120 sleep     umtxn
umtxn
54318 117468 suricata W#03        -1 120 sleep     uhwait
54318 117469 suricata W#04        -1 120 sleep     umtxn
umtxn
54318 117470 suricata W#05        -1 120 sleep     uhwait
54318 117471 suricata W#06        -1 120 sleep     uhwait
uwait
54318 117472 suricata W#07        -1 120 sleep     umtxn
54318 117473 suricata W#08        -1 120 sleep     umtxn
umtxn
54318 117474 suricata W#09        -1 120 sleep     umtxn
54318 117475 suricata W#10        -1 120 sleep     umtxn
uwait
54318 117476 suricata W#11        -1 120 sleep     umtxn
54318 117477 suricata W#12        -1 120 sleep     umtxn
umtxn
54318 117478 suricata FM#01       -1 120 sleep     nanslp
54318 117479 suricata FR#01       -1 120 sleep     nanslp
54318 117480 suricata CW          -1 120 sleep     umtxn
uwait
54318 117481 suricata CS          -1 120 sleep     uwait
uwait
54318 117482 suricata US          -1 120 sleep     select
```

```
# procstat kstack 54318
PID TID COMM    TDNAME          KSTACK
54318 104027 suricata Suricata-Main mi_switch sleepq_catch_signals sleepq_timedwait_sig_sleep p kern_cl
ock_nanosleep sys_nanosleep amd64_syscall fast_syscall_common
54318 117465 suricata RX#01-igb0 mi_switch sleepq_catch_signals sleepq_wait_sig_sleep umt xg_sleep
do_wait __umtx_op_wait_uint_private sys__umtx_op amd64_syscall fast_syscall_common
```

07/17/2022
Console output attached. ^C sent, the process stays up another minute and terminated with the error message at the end.

#6 - 07/01/2021 02:08 PM - Martin Rehak
- File debug.out.gz added

Full terminal output with suricata compiled with --enable-debug and SC_LOG_LEVEL=Debug. Let me know what other info would you like to see.

Thank you

#7 - 07/04/2021 02:13 PM - Bob Smith

Seeing the exact same issue on my FreeBSD R13.0 installation with Suricata 6.0.3

#8 - 07/09/2021 11:55 AM - Jeff Lucovsky

Could you share your netmap configuration sections from suricata.yaml?

#9 - 07/09/2021 01:31 PM - Bill Meeks
Jeff Lucovsky wrote in #note-8:

Could you share your netmap configuration sections from suricata.yaml?

Not sure which of us reporting in this issue you meant, but for completeness here is the netmap section from the suricata.yaml I used in my FreeBSD-12.2/STABLE test VM.

```yaml
netmap:
  - interface: em0
    threads: auto
    copy-mode: ips
    copy-iface: em0^
    # Set to yes to disable promiscuous mode
    disable-promisc: yes
    checksum-checks: auto
    # BPF filter to apply to this interface. The pcap filter syntax apply here.
    #bpf-filter: port 80 or udp
    - interface: em0^
      threads: auto
      copy-mode: ips
      copy-iface: em0
    # Put default values here
    - interface: default
```

To keep things simple, I only enabled the following built-in rules while testing:

```yaml
## Configure Suricata to load Suricata-Update managed rules.
##
default-rule-path: /usr/local/share/suricata/rules
rule-files:
  - decoder-events.rules
  - stream-events.rules
  - dns-events.rules
```

The VM was bridged to my real LAN. I then set up a ping test against the VM from my workstation PC and waited. It would work for some period of time (usually just a few minutes, less than 5 usually), then just quit responding.

#10 - 07/09/2021 01:57 PM - Jeff Lucovsky
Thanks Bill ... the question was intentionally left open :-) 

#11 - 07/13/2021 12:50 PM - Jeff Lucovsky
I've not been able to reproduce this issue in workers runmode.

It reproduces with autofp runmode and 2 capture threads.

Investigation continuing.

#12 - 07/17/2021 11:37 PM - Bill Meeks
I may have found a contributor to this issue. I am testing now to verify my hunch.

When the code in source-netmap.c is opening a netmap port to the Host Stack (for example, "em0^"), it fails to specify that only the TX or RX rings should be opened for a given direction. That leaves netmap to use its default behavior, which is to open BOTH the TX and RX rings of the Host Stack and assign them to the file descriptor. This can be problematic because as part of the IPS operating mode, where the Host Stack is one side of the netmap connection, Suricata needs to open two netmap file descriptors as follows:

Example below assumes NIC is using em driver in FreeBSD
Netmap file descriptor 1 = em0 -> em0^ Netmap file descriptor 2 = em0^ -> em0

Two descriptors are necessary in order for bi-directional traffic flow. That is, RX -> TX one way, and TX -> RX the other way.

However, the above means that unless the TX and RX rings are separately opened for a given interface (RX in one descriptor, and TX in the other), then you can have collision issues because these netmap file descriptors are accessed by different threads in Suricata. One or more threads perform the "read" operation, and a different packet processing thread is calling a Netmap Callback function to "write" any "not-dropped" packets back out. Both the read and write routines ultimately are also using poll(), and the default behavior of netmap with poll() is to process both RX and TX rings unless the descriptor explicitly isolated them in the initial open call.

I think the solution is to open the RX and TX Host Rings separately, and assign them to unique file descriptors as is done already with the physical
NIC rings. So you would open the physical RX rings and the Host Stack TX rings in one file descriptor; and then the physical TX rings and the Host Stack RX rings in another file descriptor. This separates them so that the poll() system call will only process the rings in the current descriptor (i.e., only the RX or TX rings as specified for each descriptor). This issue may have been masked either completely, or partially, by some of the old locking code in the Packet queue handling code that was removed in Suricata 6.x.

I have made the necessary changes to the source code, and I am currently testing in a pfSense virtual machine to see if the lock and subsequent hang condition occurs. I am also testing if this change eliminates another netmap error seen in both Suricata 5.x and 6.x. That error is: netmap_ring_reinit called for <interface_name> coupled with nm_txsync_prologue <interface_name>. According to the netmap developer, the most common cause for the error is concurrent access to the head, cur, and tail pointer variables by multiple threads. I believe this is happening in the Suricata code due to the failure to separate the RX and TX rings for the Host Stack into different file descriptors.

#13 - 07/18/2021 01:53 PM - Jeff Lucovsky
Sounds like a good direction however I wonder why it's occurring with more prevalence with 6.0.x than with 5.0.x.

#14 - 07/18/2021 04:25 PM - Bill Meeks
Jeff Lucovsky wrote in note-13:

Sounds like a good direction however I wonder why it's occurring with more prevalence with 6.0.x than with 5.0.x.

Unfortunately my optimism was shortlived. It hung about an hour after I posted my note above. I still think it is proper to open the Host TX and RX rings specifically for the descriptors the same as is done with the NIC hardware rings.

I do wonder if the changes to the Packet Queue Handling (via changes to the threading logic in tmthread.c) in Suricata 6.x is the clue, though. I was reviewing the History of changes to those source files in GitHub, and there were a number of changes there in 6.x as referenced back to 5.x. This commit, in particular, is the one I'm wondering may be related to this bug: https://github.com/OISF/suricata/commit/f8aed4ce2dd8ef141f5dd2ae64c9faccf8f0ca2#diff-e46486d647cc677301e9a361a93037b15a6dbbb0c390f76c0b2107515c8a2. It talks about removing previously "locked" PacketQueue logic in favor of using a non-locked Queue. Our failure mode is some kind of kernel lock or deadlock condition when doing packet I/O. Perhaps the changes in that commit opened a mechanism for a race condition when sending Packet structures back to netmap for disposition??

I also wonder what roll calls to the poll() system function play in this hang. The only modules I see that use a poll() system call are af-packet, source-ifpw and source-netmap. Could maybe the call to poll() result in some kind of deadlock under certain circumstances, such as two Suricata threads trying it with the same file descriptor?

#15 - 07/19/2021 07:40 AM - Martin Rehak
Seeing that the changes you have prepared relates to netmap (as far as I understand) I would like to note that I am suffering the issue without using netmap.

# suricata --pcap=igb0 -vvvv

#16 - 07/19/2021 01:39 PM - Bill Meeks
Martin Rehak wrote in note-15:

Seeing that the changes you have prepared relates to netmap (as far as I understand) I would like to note that I am suffering the issue without using netmap.

[...]

Yes, your experience indicates to me that the root cause appears to be deeper than just the netmap code. It makes me wonder if the changes in threading logic merged in the GitHub commit I referenced above are the culprit. Perhaps those changes, and changes in FreeBSD itself, don't mix well together???

I can reproduce the hang using netmap in either autofp runmode or workers runmode, although the hang appears to happen slightly sooner in autofp runmode.

#17 - 08/19/2021 02:51 PM - Bill Meeks
Time for an update on this issue. Been working offline for a while with some of the Suricata team on debugging this problem, and I think the solution has been found. Early testing has been successful. I would expect the fix to be posted into master soon.

It is likely, in my view, that this bug would have also impacted packet acquisition modes other than netmap. I believe the impact of the bug was operating system dependent, as it had to do with how mutex locking was handled. So some environments may have experienced the issue while many others did not.

#18 - 08/19/2021 03:14 PM - Victor Julien
- Assignee changed from Jeff Lucovsky to Victor Julien
- Target version changed from 6.0.4 to 7.0rc1
- Affected Versions 6.0.0, 6.0.1, 6.0.3, git master added
- Label Needs backport to 6.0 added

#19 - 08/19/2021 03:21 PM - Victor Julien
- Status changed from Assigned to In Review

https://github.com/OISF/suricata/pull/6301

#20 - 08/23/2021 07:55 PM - Victor Julien
- Status changed from In Review to Closed

https://github.com/OISF/suricata/pull/6301

#21 - 08/25/2021 12:59 PM - Jeff Lucovsky
- Copied to Bug #4629: freebsd: lockups due to mutex handling issues added

#22 - 11/18/2021 03:14 PM - Victor Julien
- Subject changed from Netmap IPS mode in Suricata 6.x ceases to pass traffic after a short, variable period of time on FreeBSD-12 to freebsd: lockups due to mutex handling issues

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>out.txt</td>
<td>25.2 KB</td>
<td>07/01/2021</td>
<td>Martin Rehak</td>
</tr>
<tr>
<td>debug.out.gz</td>
<td>24.2 MB</td>
<td>07/01/2021</td>
<td>Martin Rehak</td>
</tr>
</tbody>
</table>