Suricata - Feature #3319

on 'wrong thread' reinject packets to correct thread

11/05/2019 03:04 PM - Victor Julien

Status:	New		
Priority:	Normal		
Assignee:	OISF Dev		
Category:			
Target version:	TBD		
Effort:		**Label:**	
Difficulty:			

Description

While I'm reviewing (and of course at the same time cleaning up) the threading framework, I was thinking we could 'solve' the wrong-thread issue by reinjecting a packet into the correct thread. This would of course have a perf impact, likely a large one, but it might be a better 'failure mode' than just continuing in this known bad state. It would not be a 100% fix either, as packet order might still get messed up by this.

I suppose we could then disable further stream events for that flow, and maybe in general put it into some kind of 'liberal' state wrt stream engine checks.

We'd still increment wrong_thread counters and such, as the goal of a serious deployment should be to minimize that.

Related issues:

Related to Support #2725: stream/packet on wrong thread	**Feedback**

History

#1 - 11/05/2019 03:04 PM - Victor Julien

- Related to Support #2725: stream/packet on wrong thread added

#2 - 11/07/2019 10:00 PM - Andreas Herz

It would be also helpful to have some sort of verbose logging for that to find out WHY this even happens in solid symmetric RSS queue+cluster_qm setups. Even if it's just a dedicated compile flag like profiling. But without rollover and flow distribution either by sd or sdfn flag I would not expect it to happen but it does sometimes. So in the end this might even help using a nice logic to reinject them.

#3 - 11/09/2019 09:29 AM - Victor Julien

Andreas Herz wrote:

> It would be also helpful to have some sort of verbose logging for that to find out WHY this even happens in solid symmetric RSS queue+cluster_qm setups. Even if it's just a dedicated compile flag like profiling. But without rollover and flow distribution either by sd or sdfn flag I would not expect it to happen but it does sometimes. So in the end this might even help using a nice logic to reinject them.

Not sure I understand. How would Suricata know why this happens? ATM we can't even figure out manually why this happens in many cases, so I see no way to have Suricata somehow magically know this. We could consider building a script of some kind to check for common mistakes in the system setting that can cause this.

I think this discussion should have its own ticket or be part of #2725 btw. This ticket is about whether Suricata should try to correct the case when it encounters it.

#4 - 02/25/2020 04:03 PM - Cooper Nelson

Andreas Herz wrote in #note-2:

> It would be also helpful to have some sort of verbose logging for that to find out WHY this even happens in solid symmetric RSS queue+cluster_qm setups. Even if it's just a dedicated compile flag like profiling. But without rollover and flow distribution either by sd or sdfn flag I would not expect it to happen but it does sometimes. So in the end this might even help using a nice logic to reinject them.

Peter Manev observed that switching to a trivial 'sd' hash eliminated the problem in his test case, so at this point I'm fairly certain what we are observing is simply an artifact of how RSS is implemented. So we need to account for that in some way, either forcing a trivial hash or modifying suricata to properly handle these edge cases. If the clustering modes completely ignored both the RSS hash and queue and computed an entirely new hash for each packet after reassembly this should fix the issue for most cases. The clustering modes other than 'flow' would probably have to be

modified so they properly copy the packet to a different queue when necessary.

Thinking about it, there is probably an edge case where an IP reassembly fails due to one of the fragments being dropped (which is common). Clients and servers will just drop these packets and increment an internal 'reassembly failed' counter, however AF_PACKET will simply hand off the fragment to suricata for processing. So even if you are doing proper 'sdfn' hashing of reassembled packets, the partial fragment will still end up on the wrong thread and increment the counter. The solution here would be to add an option for suricata to drop reassbly fails, much like the option to drop packets with bad checksums.

#5 - 02/25/2020 08:58 PM - Cooper Nelson

Actually, thinking about I think this could work with minimal performance impact or packet re-ordering if the check was done prior to copying the packet to the worker thread.

It would probably take some experimenting to see whether its cheaper in terms of cpu cycles to do a check before the copy, vs. just generating a new hash for each packet.