Suricata - Feature #2958
Suricata 5.0.0beta1 and way too much anomaly logging
05/03/2019 09:31 AM - Andre ten Bohmer

<table>
<thead>
<tr>
<th>Status:</th>
<th>Assigned</th>
<th>Priority:</th>
<th>Normal</th>
<th>Assignee:</th>
<th>Jeff Lucovsky</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td></td>
<td>Target version:</td>
<td>7.0beta1</td>
<td>Effort:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

If outputs: -> -eve-log: -> types: -> - anomaly: is enabled in suricata.yaml, eve.json gets flooded with event type anomaly. I've seen more than 13 million of these in 5 minutes which also drastically reduces performance seen capture.kernel_drops.

capture.kernel_drops was under v4.1.3 way below 0.01% and now I see numbers like:

capture.kernel_packets | Total | 47542250
|-----------------------|-------|----------------|
capture.kernel_drops   | Total | 37202776

Event logged in eve.json:

{ "timestamp": "2019-05-03T09:11:57.277701+0200", "in_iface": "ens2f0", "event_type": "anomaly", "vlan": [403], "anomaly": { "type": "packet", "event": "decoder.ipv4.trunc_pkt" }}

{ "timestamp": "2019-05-03T09:11:55.623627+0200", "in_iface": "ens2f1", "event_type": "anomaly", "vlan": [403], "anomaly": { "type": "packet", "event": "decoder.ipv4.trunc_pkt" }}

Is it possible to limit this logging? An other option/solution?

TIA!

**History**

#1 - 05/07/2019 01:04 PM - Victor Julien

For now [https://github.com/OISF/suricata/pull/3829](https://github.com/OISF/suricata/pull/3829) has been merged. It disables the log by default and adds a warning to the yaml we ship. We'll be working on this further.

#2 - 05/11/2019 04:57 PM - Jeff Lucovsky

- Assignee set to Jeff Lucovsky

Here are some possible directions for reducing anomaly log activity:

**Options:**

- Rate limit log records. Use a mechanism like the Linux kernel's "printk ratelimit" that restricts the number of messages logged within a time interval. Log records that exceed the threshold are dropped; when drops occur, a log record is logged stating how many records were dropped when the next log record is written. The advantage of this approach is simplicity; the disadvantage is lost records.

- Store and forward. Batch successive log records into a fixed size memory area (size TBD). When memory area reaches capacity, the accumulated logs will be written. This maintains ordering at the expense of latency. An option would be to buffer messages until (1) a time threshold or (2) a size/count threshold is reached. Which ever occurs first, causes the log records to be written. This approach increases the memory footprint of Suricata but amortizes the cost to write over many records. This approach is simple, doesn't lose information, smooths jitter but uses more memory.

- Compress adjacent like records. Adjacent log records that are the same* (sameness tbd) would be accumulated with and marked with an occurrence value. This approach will store the last record that would've been logged and increase it's occurrence count as long as subsequent records are identical (TBD). When a non-identical record is submitted, the record being held is logged (output) and the non-identical record is buffered as long as subsequent records are identical. This is like the store and forward mechanism with store size of 1 and a semantic that identical records will be combined and the duplicates discarded. The chief disadvantage is complexity and performance may continue to suffer when few successive records are deemed identical.

- Filtering options. The chief drawback is that no relief may be provided when the filter choice doesn't match or isn't suitable for the anomalies that are occurring. Some ideas on filter choices:
  - Filter on layer 3 protocol (unable to determine, ip, icmp)
  - Filter on layer 4 protocol (udp, tcp, ...).
  - Filter on layer 7 protocol (if available).
  - Filter on whether packet is invalid (PKT_IS_INVALID) or not.
Filter on specific decode events. This would be difficult to explain and configure.
A combination of one or more of the preceding choices.

#3 - 07/02/2019 10:12 PM - Jeff Lucovsky
We will be working to mitigate log volume by extending the anomaly configuration with the following toggles; each toggle individually enables/disables logging of the related events; each toggle value is ignored if anomaly logging is disabled.
Proposed toggles:
- Logging of protocol parser events
- Logging of parser related events
- Logging or protocol detection related events
- Logging of packet decode related events

#4 - 07/05/2019 08:50 PM - Andreas Herz
- Target version set to TBD

#5 - 09/24/2019 10:10 AM - Victor Julien
- Status changed from New to Assigned
- Priority changed from Low to Normal
- Target version changed from TBD to 6.0.0beta1

I think all or most suggestions from https://redmine.openinfosecfoundation.org/issues/2958#note-3 have been implemented. It would be nice to consider more advanced filters as well in this ticket.

#6 - 05/25/2020 03:49 PM - Victor Julien
- Target version changed from 6.0.0beta1 to 7.0beta1

#7 - 01/02/2021 03:56 PM - Jeff Lucovsky
Perhaps we could add
- Layer 3 protocol filter, e.g.,
  
  net_proto=!IP

- Layer 4 protocol filter:
  
  proto=UDP
  or
  proto=!TCP

- Layer 7 protocol filter:
  
  app_proto=HTTP
  or
  app_proto=[SNMP, SMB]

Thoughts?