Suricata - Feature #197

Investigate removing memcpy() from pcap runmodes and possibly others.

06/29/2010 06:23 PM - Will Metcalf

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Eric Leblond</td>
</tr>
<tr>
<td>Category:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>1.4beta1</td>
</tr>
<tr>
<td>Effort:</td>
<td></td>
</tr>
<tr>
<td>Difficulty:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

currently in the pcap run modes inside of the callback function we do the following memcpy()

memset(p->pkt, pkt, p->pktlen);

This is an artifact from the NFQUEUE and is needed by that run mode, but may not be needed by libpcap. We should investigate modifying pkt to simply use the ptr given to use by libpcap. Not sure if this is thread safe etc. In my initial test making the following code modifications lead to a 20% increase but also a pretty big loss in accuracy.

```c
#include <uint8_t pkt[IPV6_HEADER_LEN + 65536 + 28];
uint8_t *pkt;

source-pcap.c
//memset(p->pkt, pkt, p->pktlen);
p->pkt = pkt;
```

**History**

#1 - 06/30/2010 01:35 AM - Victor Julien

Definitely worth looking at. It's probably failing now because of the defrag engine needing that buffer for copying the reassembled packet into.

#2 - 06/30/2010 09:34 AM - Jason Ish

Defrag should only need to be modified with respect to storing the reassembled, so a pretty minor fix there.

But will this work? This buffer that is passed to us is managed by libpcap, and will be reclaimed by libpcap at its will. Is there a way to tell libpcap not to reclaim a buffer until we're ready to let it go?

I did a simple test app that made a copy of the packet and stored a pointer to it. The both contained the same data only for about 32 packets (just a very default settings pcap app). Increasing the pcap buffer size may push this out a bit, but its still not under our control.

If you look at pcap_next_ex, which is kind of a wrapper around pcap_dispatch so you can avoid using the callback yourself, it places the packet in a buffer you provided. So you still have the memcpy, but this time inside libpcap rather than in your app. This is required because once the callback has returned you can't rely on the contents of the packet buffer passed to you inside the callback.

I have to admit I haven't kept fully up to date on the latest libpcap developments so there may be away around this.

#3 - 06/30/2010 02:17 PM - Jason MacLulich

This is the same problem we encounter with DAG integration -- we have to memcpy the packets from the DAG buffers as the DAG card writes to the same ring buffer.

We do have control though on when the DAG card can write over packets but this information would have to be propagated through the engine in some fashion, you would still require copying pkts from the buffer for IP/TCP reassembly, but you may avoid an initial copy.

In the past with more specialized applications I've been able to keep packets in the buffer and then copy them out when we've needed to, e.g. IMA (Inverse Multiplexing over ATM) lends itself nicely to this -- this is a relatively simple application though compared with Suricata.

Could be interesting work to see if this is applicable -- Will what sort of HW did you do your testing on?

#4 - 07/01/2010 07:42 AM - Will Metcalf

"If you look at pcap_next_ex, which is kind of a wrapper around pcap_dispatch so you can avoid using the callback yourself, it places the packet in a
buffer you provided. So you still have the memcpy, but this time inside libpcap rather than in your app.

Does this buy us anything performance wise?

#5 - 09/14/2011 05:47 AM - Victor Julien
PF_RING now supports this and I think work is being done to support this for netfilter_queue as well.

#6 - 06/14/2012 06:41 AM - Victor Julien
- Status changed from New to Assigned
- Assignee changed from Victor Julien to Eric Leblond

Eric, can you figure out if there is a way to remove the memcpy for libpcap mode?

#7 - 06/14/2012 11:17 AM - Eric Leblond
After a first read, it seems we can use the zero copy under the same assumption as with af_packet: all treatment must be made within the callback thread. This means only 'single' and 'workers' mode can benefit from this. I preparing a patch to implement this.

#8 - 06/14/2012 12:06 PM - Eric Leblond
- File 0001-pcap-enable-zero-copy-mode-in-some-running-mode.patch added
- % Done changed from 0 to 80

The attached patch brings zero-copy to 'workers' and 'single' mode.

#9 - 06/15/2012 05:44 AM - Victor Julien
Maybe you can address #415 as well, so that we don't have to limit ourselves to single and workers? Or does pcap not have a "release packet" function for Ocopy?

#10 - 06/15/2012 08:12 AM - Eric Leblond
pcap does not have such a function but I start working on #415 for other capture mode.

#11 - 07/04/2012 03:24 AM - Eric Leblond
- Target version changed from TBD to 1.4beta1

#12 - 09/05/2012 09:04 AM - Eric Leblond
This can be closed as #415 implement a release data mechanism.

#13 - 09/05/2012 09:31 AM - Victor Julien
- Status changed from Assigned to Closed
- % Done changed from 80 to 100

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-pcap-enable-zero-copy-mode-in-some-running-mode.patch</td>
<td>2.47 KB</td>
<td>06/14/2012</td>
</tr>
</tbody>
</table>