### Suricata - Optimization #3322

**Use standard CRC32 for hash-like functions**

11/06/2019 09:18 AM - Philippe Antoine

<table>
<thead>
<tr>
<th>Status:</th>
<th>In Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Philippe Antoine</td>
</tr>
<tr>
<td>Category:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>TBD</td>
</tr>
<tr>
<td>Effort:</td>
<td>low</td>
</tr>
<tr>
<td>Difficulty:</td>
<td></td>
</tr>
</tbody>
</table>

**Label:**

**Description**

Instead of a custom one (as CRC32 was I think designed for kind of avoiding collisions)

One such function is StringHash cf [https://github.com/OISF/suricata/pull/4337](https://github.com/OISF/suricata/pull/4337)

---

### History

#### #1 - 11/06/2019 11:30 AM - Victor Julien

As we discussed offline, it would be nice to create some kind of benchmarking framework where we could validate such changes. Pure pcap tests may not always give enough insight. For example with the bm optimizations pcap based tests showed no difference, while I think more micro level benchmarks would have shown something.

#### #2 - 11/06/2019 12:58 PM - Philippe Antoine

It would be nice if this benchmarking framework handles caches realistically.

With the example of Boyer-Moore optimizations (one less call to alloc), I am not sure a naive benchmarking would shows much difference as the additional call to alloc would grab repeatedly the same cached memory area, whereas in a real Suricata execution, this would not be the case.

#### #3 - 11/07/2019 10:01 PM - Andreas Herz

- Assignee set to Philippe Antoine
- Target version set to TBD

#### #4 - 11/13/2019 09:21 AM - Philippe Antoine

MurmurHash may be the best function [https://en.wikipedia.org/wiki/MurmurHash](https://en.wikipedia.org/wiki/MurmurHash)

Current function is not random
It is DJB hash [https://stackoverflow.com/questions/10696223/reason-for-5381-number-in-djb-hash-function](https://stackoverflow.com/questions/10696223/reason-for-5381-number-in-djb-hash-function)

Here is a comparison [https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed](https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed)

#### #5 - 11/15/2019 09:30 AM - Philippe Antoine

I put a first benchmark here [https://github.com/OISF/suricata/pull/4371](https://github.com/OISF/suricata/pull/4371)

#### #6 - 02/18/2020 12:53 PM - Philippe Antoine

- Status changed from New to In Review

[https://github.com/OISF/suricata/pull/4371](https://github.com/OISF/suricata/pull/4371)

#### #7 - 05/21/2020 05:42 AM - Roland Fischer

There are a few other hash functions that might be interesting. Google's CityHash comes to mind as it's their "general purpose" hash. SpookyHash/xxhash might be options as well. The stackexchange page lists them as well I think.

Ultimately, it depends on how much time you want to spend on this vs how happy you are with the current hash. Plus, what the usage patterns of the hash are as well as the data to be hashed. ;)

06/13/2020