## Suricata - Feature #3317
### rules: use rust for tokenizing rules

11/05/2019 12:07 PM - Victor Julien

<table>
<thead>
<tr>
<th>Status:</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Jason Ish</td>
</tr>
<tr>
<td>Category:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>TBD</td>
</tr>
<tr>
<td>Effort:</td>
<td></td>
</tr>
<tr>
<td>Difficulty:</td>
<td></td>
</tr>
</tbody>
</table>

### Description
Idea is having a Rust language rule tokenizer that lives in our code base, but is also pushed a standalone crate.

### Related issues:
- Related to Task #3195: tracking: rustify all input
- Related to Bug #1926: rule parsing: wrong content checked for fast_pattern (snort compatibility)

### History

**#1 - 11/05/2019 12:07 PM - Victor Julien**
- Related to Task #3195: tracking: rustify all input added

**#2 - 11/05/2019 12:11 PM - Victor Julien**
- Subject changed from rules: use rule for tokenizing rules to rules: use rust for tokenizing rules

**#3 - 11/05/2019 12:40 PM - Victor Julien**
- Related to Bug #1926: rule parsing: wrong content checked for fast_pattern (snort compatibility) added

**#4 - 11/06/2019 11:31 AM - Victor Julien**

> It would be nice if this could replace much of our pcre use in parsing. The pcre use is both for tokenizing and input validation. The tokenizing works well, but the input validation less so. Its hard to produce clear errors that are better than "the regex said no".

If the rust based code does all the tokenizing, we'd need more input value validation to make up for it.

**#5 - 11/06/2019 11:31 AM - Victor Julien**
(by Jason Ish. Moved from #3195)

Victor Julien wrote:

> It would be nice if this could replace much of our pcre use in parsing. The pcre use is both for tokenizing and input validation. The tokenizing works well, but the input validation less so. Its hard to produce clear errors that are better than "the regex said no".

> If the rust based code does all the tokenizing, we'd need more input value validation to make up for it.

The way I see is it the top level parser will give you a tuple of (keyword, value), but will not have done any validation of that value to make sure its correct for that keyword. It will be up to the handler for that keyword to parse that like it is now. So it would get rid of pcre in this outer tokenizer, but the parsing of the values would be on a keyword by keyword basis.

**#6 - 11/06/2019 11:35 AM - Victor Julien**

I think this highest level tokenizing is already done w/o pcre since some time.

Not sure I see much value in a rust crate that would just do the highest level of tokenizing. I was more thinking about having a rule parser that could be the single source of 'truth' for Suricata rule parsing and validation. This mean it would have to be much more aware of the individual keywords and their syntax. Maybe this isn't feasible.

**#7 - 11/06/2019 02:06 PM - Jason Ish**

Its a mix of effort and re-usability I think.
A tokenizer would satisfy requirements of preprocessing rules - such as Suricata-Update, or basic enable/disables.

You typically might tokenize, then pass off to the parser. We could implement that as a standalone module, but is a lot more work as it has to understand every keyword. You'd want to parse all the values into some struct that could then be use by that keyword implementation, so it doesn't need to reparse the value again. We could have a generic one for keywords that are unknown to the parser, which would allow us to implement keywords over time.

But I see it as a tokenizer, which would then feed to a parser.

#8 - 11/06/2019 06:54 PM - Victor Julien

Ok, I guess I see little to no value in just a simple high level tokenizer in Rust. The current code is fast and simple, so we wouldn't gain much.