Suricata - Task #3307
Research: evaluate future of lua support in Suricata
11/02/2019 09:24 AM - Victor Julien

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

**Description**
Suricata has had support for Lua for a long time. Originally we only supported luajit, but later this was extended to regular lua. Luajit was giving us lots of issues and some design limitations. Due to luajit targeting lua 5.1, we've worked with that for the regular lua as well. Lua 5.1 is starting to be considered legacy, and recent versions of distributions are not including it anymore.

One problem that we never really considered is that lua versions include breaking changes. So it is tricky to support all versions of lua in a single script (or maybe impossible). This means that a generic way to ship lua rules, for example as part of a ruleset, is also going to be difficult or even impossible. Unless we are willing to drop support for luajit, we can't move away from the lua 5.1 'standard'.

It looks like one of the main use cases for lua is gaming engines and it seems that these generally vendor lua in, so that they have a single version to work with. We could consider something like this. It would bring overhead and I could imagine distro's not being excited about this.

A drawback of lua is the fragile ecosystem in which many modules exist, but quality of them varies wildly. Also various methods for installing those modules exist. For new users this is quite confusing. Despite these drawbacks the modules have made some really interesting script possible, either for real time inspection or more focused at offline pcap analysis.

At Suricon 2019 it was suggested that perhaps a more modern technology like WASM would be a better fit for us.

**Related issues:**
- Related to Task #3288: Suricon 2019 brainstorm
- Related to Task #3329: Research: WASM as a Lua alternative and for dynamically loadable modules added
- Related to Task #4097: Suricon 2020 brainstorm
- Related to Feature #1971: lua: make mandatory
- Related to Task #4762: Suricon 2021 brainstorm
- Related to Feature #4775: lua: overhaul lua support

**History**

#1 - 11/05/2019 10:49 AM - Victor Julien
- Parent task deleted (#3288)

#2 - 11/05/2019 10:49 AM - Victor Julien
- Related to Task #3288: Suricon 2019 brainstorm added

#3 - 11/05/2019 11:25 AM - Victor Julien
- Assignee set to OISF Dev
- Target version set to TBD

#4 - 11/09/2019 09:13 AM - Victor Julien
- Related to Task #3329: Research: WASM as a Lua alternative and for dynamically loadable modules added

#5 - 11/12/2019 04:07 PM - Victor Julien
When considering the future of our lua integration, it might be interesting to keep an eye on https://github.com/dibyendumajumdar/ravi

03/09/2022 1/2
From Suricat 2020 brainstorm: Python suggested as a scripting language.

One new point that was brought up is that lua has been successfully sandboxed in other tools, so we could at least improve that side of things.

Sascha posted the link [https://github.com/dbohdan/embedded-scripting-languages](https://github.com/dbohdan/embedded-scripting-languages), which has a lot of options for embedding a scripting language. Clearly we should no go down the path of creating our own with so many options out there. But we should also think about the ecosystem of the language. We could choose something that is just a language, with no libraries, etc. Suricata has to export all functionality into the script runtime, and a library base could be built up from users. This is more or less what Zeek has to do.

With Lua, for better or worse you can have access to the Lua ecosystem (Luarocks, etc). Likewise for Javascript, Python, etc.

Jo did bring up that Lua can be sandboxed and argued that Lua is almost perfect for what we need it for (the most common use of Lua to extend rules), which might be true, but only true if we can lock to specific version, which may mean vendoring. Or ending up in a situation where its not available everywhere if a distribution drops support for the version we lock to.

One issue with vendoring in a version is that if the runtime is no longer supported upstream we will have to support it for security fixes and general bugs as well. I find this a compelling argument for choosing something that is still maintained. This then means you commit to a release cadence of that project. So then I think a framework that cares about backwards compatibility would be strongly preferred.

Jo did bring up that Lua can be sandboxed and argued that Lua is almost perfect for what we need it for (the most common use of Lua to extend rules), which might be true, but only true if we can lock to specific version, which may mean vendoring. Or ending up in a situation where its not available everywhere if a distribution drops support for the version we lock to.

One issue with vendoring in a version is that if the runtime is no longer supported upstream we will have to support it for security fixes and general bugs as well. I find this a compelling argument for choosing something that is still maintained. This then means you commit to a release cadence of that project. So then I think a framework that cares about backwards compatibility would be strongly preferred.

After more discussion at Suricon 2021 I'm starting to lean towards to sticking with lua, however overhauling it completely:

- vendor a specific version in (probably the highest current stable), so rule writers can target a specific version
- disallow modules by default (or at all). This would mean Suricata needs to expose more functionality
- do proper sandboxing, both for security and performance. This would mean Suricata needs to expose more functionality
- enable it by default so rule vendors can rely on it to be there

We can then bump the vendored major version on Suricata's major version updates. This might break compatibility of existing scripts, but this is not unlike changes we make to other parts of Suricat's input and output.