YAML: probably not so great after all

Written on 4 Sep 2016 – last updated on 27 Dec 2017.

I previously wrote why using JSON for human-editable configuration files is a bad idea. Today we're going to look at some general problems with the YAML format.

Insecure by default

YAML is insecure by default. Loading a user-provided (untrusted) YAML string needs careful consideration.

```
!!python/object/apply:os.system
args: ['ls /']
```

Running it with `print(yaml.load(open('a.yaml')))` should give you something like:

```
bin   etc   lib    lost+found  opt   root  sbin  tmp  var sys
boot  dev   efi    home        lib64 mnt   proc  run  srv usr
0
```

Many other languages (including Ruby and PHP[1]) are also unsafe by default. Searching for `yaml.load` on GitHub gives a whopping 2.8 million results. `yaml.safe_load` only gives 26,000 results.

Mind you, many of those `yaml.load()`s are fine – loading in a config file with `yaml.load()` is often okay since it's usually (though not always!) from a 'trusted source', and many are from test files with static YAML. But still, one can't help but wonder how many exploits are hidden in those 2.8 million results.

This is not a theoretical problem. In 2013 every Ruby on Rails application ever written was found to be vulnerable to remote code execution due to exactly this problem.

One might argue this is not really the fault of the YAML format *as such*, but rather the fault of the libraries implementing it wrong, but it seems to be the case that the majority of libraries are unsafe by default (especially the dynamic languages), so *de-facto* it is a problem with YAML.

One might also argue that fixing it is as easy as replacing `load()` with `safe_load()`, but many people are unaware of the problem, and even *if* you're aware of it, it's one of those things that can be easy to forget. It's pretty bad API design.

Can be hard to edit, especially for large files

YAML files can be hard to edit, and this difficulty grows fast as the file gets larger.

A good example of this are Ruby on Rails' translation files; for example:

```
en:
   formtastic:
     labels:
       title: "Title"  # Default global value
       article:
         body: "Article content"
       post:
         new:
           title: "Choose a title..."
           body: "Write something..."
         edit:
           title: "Edit title"
           body: "Edit body"
```

This still looks okay, right? But what if this file has 100 lines? Or 1,000 lines? It is difficult to see "where" in the file you are because it may be off the screen. You'll need to scroll up, but then you need to keep track of the indentation, which can be pretty hard even with indentation guides, especially since 2-space indentation is the norm and tab indentation is forbidden[2].

And accidentally getting the indentation wrong often isn't an error; it will often just deserialize to something you didn't intend. Happy debugging!

I've been happily programming Python for over a decade, so I'm used to significant whitespace, but sometimes I'm still struggling with YAML. In Python the drawbacks and loss of clarity are contained by not having functions that are several pages long, but data or configuration files have no such natural limits to their length.

For small files this is not a problem; but it really doesn't scale well to larger files, especially not if you want to edit them later on.

It's pretty complex

YAML may seem 'simple' and 'obvious' when glancing at a basic example, but turns out it's not. The YAML spec is 23,449 words; for comparison, TOML is 3,339 words, JSON is 1,969 words, and XML is 20,603 words.

Who among us have read all that? Who among us have read and *understood* all of that? Who among of have read, *understood*, and **remembered** all of that?

For example did you know there are *nine* ways to write a multi-line string in YAML with subtly different behaviour?

Yeah :-/

That post gets even more interesting if you look at its revision history, as the author of the post discovers more and more ways to do this and more of the subtleties involved.

It's telling that the YAML spec starts with a preview, which states (emphases mine):

> This section provides a quick glimpse into the expressive power of YAML. **It is not expected that the first-time reader grok all of the examples**. Rather, these selections are used as motivation for the remainder of the specification.

Surprising behaviour

What does this parse to (examples courtesy of Colm O'Connor):

```
- Don Corleone: Do you have faith in my judgment?
- Clemenza: Yes
- Don Corleone: Do I have your loyalty?
```

Yup!

```
[
    {'Don Corleone': 'Do you have faith in my judgment?'},
    {'Clemenza': True},
    {'Don Corleone': 'Do I have your loyalty?'}
]
```

Or what about:

```
python: 3.5.3
postgres: 9.3
```

`3.5.3` gets recognized as as string, but `9.3` gets recognized as a number instead of a string:

```
{'python': '3.5.3', 'postgres': 9.3}
```

Or what about:

```
Effenaar: Eindhoven
013: Tilburg
```

013 is a popular music Venue in Tilburg, but YAML will send you the wrong way since it's parsed as an octal number:

```
{11: 'Tilburg', 'Effenaar': 'Eindhoven'}
```

All of this – and more – is why many experienced YAMLers will often quote all strings, even when it's not strictly required. Many people don't use quotes, and it can be easy to forget especially if the rest of the file – possibly written by other people – doesn't use quotes.

It's not portable

Because it's so complex, its claims of portability have been greatly exaggerated. For example consider this example taken from the YAML spec:

```
? - Detroit Tigers
  - Chicago cubs
:
  - 2001-07-23

? [ New York Yankees,
    Atlanta Braves ]
: [ 2001-07-02, 2001-08-12,
    2001-08-14 ]
```

Aside from the fact that most readers of this probably won't even know what this does, try parsing it in Python with PyYAML:

```
yaml.constructor.ConstructorError: while constructing a mapping
  in "a.yaml", line 1, column 1
found unhashable key
  in "a.yaml", line 1, column 3
```

In Ruby it works:

```
{
    ["Detroit Tigers", "Chicago cubs"] => [
        #<Date: 2001-07-23 ((2452114j,0s,0n),+0s,2299161j)>
    ],
    ["New York Yankees", "Atlanta Braves"] => [
        #<Date: 2001-07-02 ((2452093j,0s,0n),+0s,2299161j)>,
        #<Date: 2001-08-12 ((2452134j,0s,0n),+0s,2299161j)>,
        #<Date: 2001-08-14 ((2452136j,0s,0n),+0s,2299161j)>
    ]
}
```

The reason for this is because you can't use a list as a dict key in Python:

```
>>> {['a']: 'zxc'}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: unhashable type: 'list'
```

And this restriction is not unique to Python; common languages such as PHP, JavaScript, and Go all share this restriction.

So use this in a YAML file, and you won't be able to read it in most languages.

Here's another example again taken from the examples section of the YAML spec:

```
# Ranking of 1998 home runs
---
- Mark McGwire
- Sammy Sosa
- Ken Griffey

# Team ranking
---
- Chicago Cubs
- St Louis Cardinals
```

Python says:

```
yaml.composer.ComposerError: expected a single document in the stream
  in "a.yaml", line 3, column 1
but found another document
  in "a.yaml", line 8, column 1
```

While Ruby outputs:

```
["Mark McGwire", "Sammy Sosa", "Ken Griffey"]
```

The reason for this is that there are multiple YAML documents in a single file (`---` start the document). In Python there is the `load_all()` function to parse all documents. Ruby's `load()` just loads the first document, and as near as I can tell, doesn't have a way to load multiple documents.

There are many more incompatibilities between implementations.

Goals achieved?

The spec states:

> The design goals for YAML are, in decreasing priority:
>
> 1. YAML is easily readable by humans.
> 2. YAML data is portable between programming languages.
> 3. YAML matches the native data structures of agile languages.

> 4. YAML has a consistent model to support generic tools.
> 5. YAML supports one-pass processing.
> 6. YAML is expressive and extensible.
> 7. YAML is easy to implement and use.

So how well does it do?

> YAML is easily readable by humans.

True only if you stick to a small subset. The full set is complex – much *more* so than XML or JSON.

> YAML data is portable between programming languages.

Not really true, as it's too easy to create constructs that are not supported by common languages.

> YAML matches the native data structures of agile languages.

See above. Plus, why only support agile (or dynamic) languages? What about other languages?

> YAML has a consistent model to support generic tools.

I am not even sure what this means and I can't find any elaboration.

> YAML supports one-pass processing.

I'll take their word for it.

> YAML is expressive and extensible.

Well, it is, but it's *too* expressive (e.g. too complex).

> YAML is easy to implement and use.

```
$ cat `ls -1 ~/gocode/src/github.com/go-yaml/yaml/*.go | grep -v _test
9247

$ cat /usr/lib/python3.5/site-packages/yaml/*.py | wc -l
5713
```

Conclusion

Don’t get me wrong, it’s not like YAML is absolutely terrible – it’s certainly not as problematic as using JSON – but it’s not exactly great either. There are some drawbacks and surprises that are not at all obvious at first, and there are a number of better alternatives such as TOML and other more specialized formats.

Personally, I’m not likely to use it again when I’ve got a choice.

If you *must* use YAML then I recommend you use StrictYAML, which removes some (though not all) of the more hairy parts.

Footnotes

1. In PHP you need to modify an INI setting for the safe behaviour; you can’t just call something like `yaml_safe()`. The PHP folks managed to make something stupid *even more stupid*. Congratulations. ↩

2. Don’t want to start the spaces vs. tabs debate here, but if tabs would be allowed I would be able to (temporarily) increase the tab width to a higher number to make it easier to see – this is sort of the point of tabs. ↩

Feedback

You can mail me at martin@arp242.net or create a GitHub issue for feedback, questions, etc.