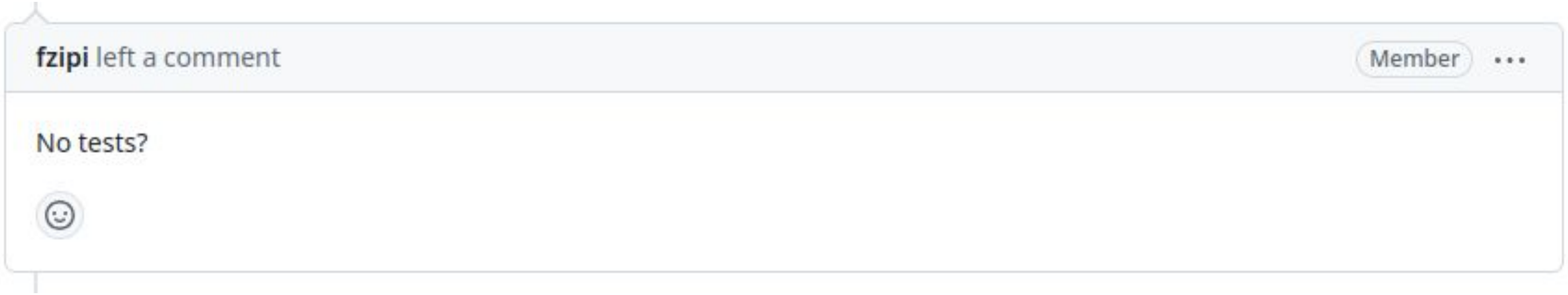
Testing ModSecurity

Why and how?

Testing ModSecurity

Reasons



fzipi left a comment

Member

No tests?

Of course, we need tests to prove that the products' behavior meets our expectations. Tests help us to check these behaviors haven't changed after any modification (CI workflow).

Now let's see what options do we have.

Review current possibilities

libmodsecurity3

- own regression test framework, part of the source tree
- uses JSON files
    - describes input (headers, body)
    - rules (including `SecRuleEngine`)
    - describe expected result
        - intervention's result
        - debug.log/error.log/parser error/...
- tests only the library
    - a few deviations from real use
        - sequence escapes in quoted strings, unmodified input (eg. REQUEST_URI)

Review current possibilities

mod_security2

- own make check / make test (part of the source tree)
  - runs tests only against operators and transformations (unit tests)
  - uses JSON files
  - useful, runs perfectly
- own make test-regression
  - written in Perl
  - slow
  - about 10% of tests are broken
  - this test case is definitely abandoned
- first view gave negative impacts
  - components were unmaintained (PCRE2 wasn't implemented), terminated the test after 30 minutes...

Review current possibilities

ModSeurity-nginx connector

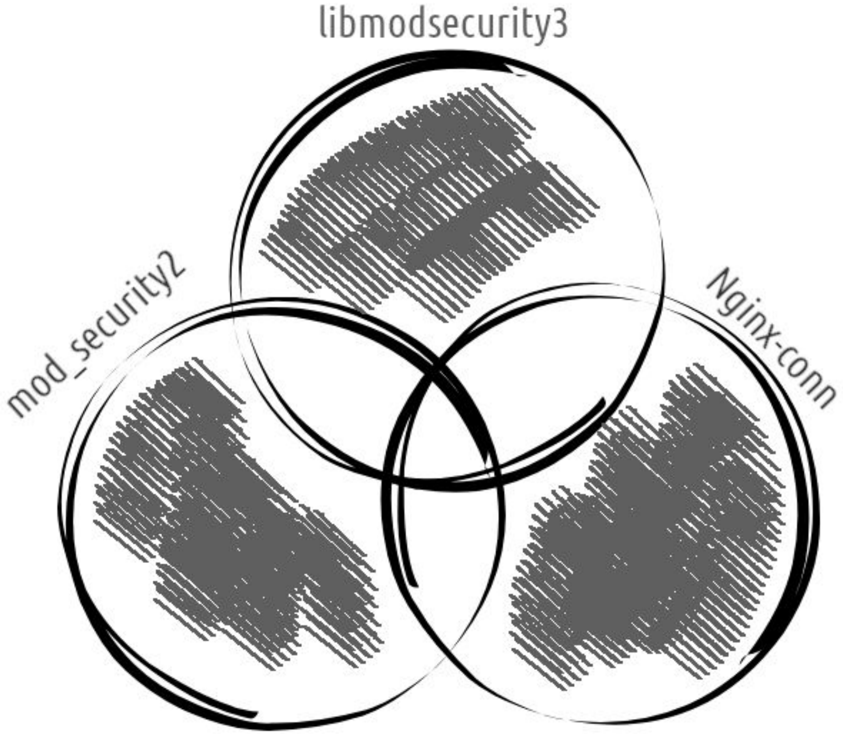
- uses Nginx's test framework
  - https://github.com/nginx/nginx-tests
  - written in Perl
  - tests are in Perl too
  - depends on Nginx (need to build)
- problem:
  - Nginx ndk exists - no need to build the whole server
  - how can a user run the tests?
  - no log - why is the test failed?

Review current possibilities

Problem

There is no intersection regarding the objectives of the tests.

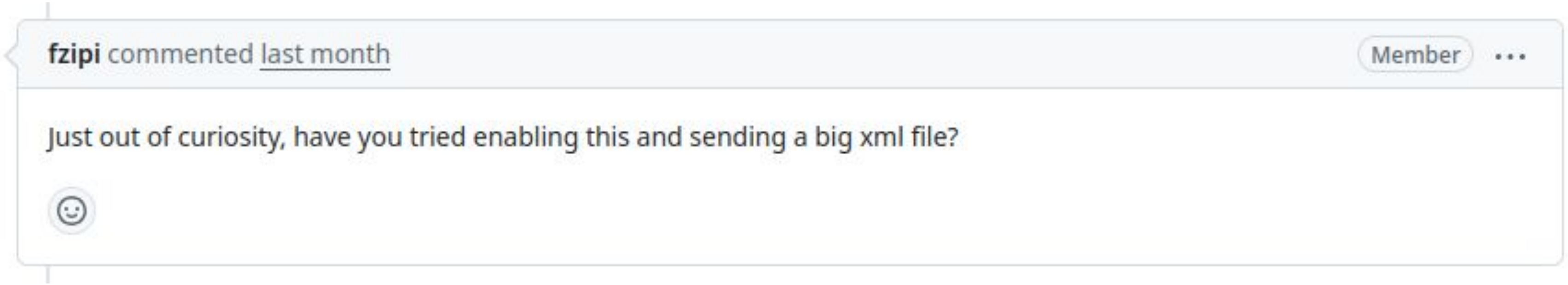
At all.



Expectations

What do we expect from the tests?

Last year experiences helped a lot...



fzipi commented last month

Member

Just out of curiosity, have you tried enabling this and sending a big xml file?

This made me think about a few questions.

Expectations

What do we expect from the tests?

- tests can be run locally without extra dependencies
- a log should be created of the tests, so that it is possible to trace why the test fails
- different payloads (urlencoded, JSON, XML)
  - unification of these
    - diff. payloads produces (almost the) same arguments
- content
- number of arguments
- size / depth

Expectations

Can we measure the results of the tests?

Probably we can measure them, but makes no sense.

Can we compare test results?

We can but also makes no sense - what guarantees the same conditions?

Main goal

Create a common test framework, not just for components above, but general.

MRTS

MRTS - last year I introduced

- developed for **web servers**
  - can't use for library
  - uses Albedo and go-ftw
- can build rules and tests for rules in one step
- can run comprehensive tests
  - eg: check a variable in all phases (diffs of `REQUEST_URI`!)
- current state
  - only few arguments are tested
  - we follow CRS - used arguments are more important

MRTS

MRTS - last year's development

Marc Stern involved a university student, Sébastien Amelinckx. He is an MSc IT student in Leuven, Belgium.

He added lot of new feature to MRTS.

He also helped in go-ftw - MRTS uses it.

His MSc thesis is about testing WAF's.

(He asked me to participate his public defense as a jury member)

MRTS

Further developments

- custom actions for templates (allow defining actions per test cases with different combinations)
- custom directives for templates (similar as above)
- extend check methods (overwriting the default output structure)
- infrastructure and test environments (build check environment)
- reusable constants (global and local constants)
- introduced complex separator
- directives for before and after rule generation

MRTS

Other developments

Based on the expectations above I started working on a new project, a payload generator.

Idea:

- define data structures
- even for multiple levels (if necessary)
- set number of iterations, parent-child relations
- generate payloads with same content in different format: JSON, XML, URLENCODED
    - for XML attributes, use "@" annotation

MRTS

More ideas

For payload generator:

- use corpus files for content data
  - like in case of CRS quantitative tests

For **MRTS**:

- use metadata to mark generated rules (tags?)
- helps to declare a minimal set of success rules
  - **#project-wafec?**

Thank you for your attention