CloudFlare’s Lua WAF

John Graham-Cumming
October 2014

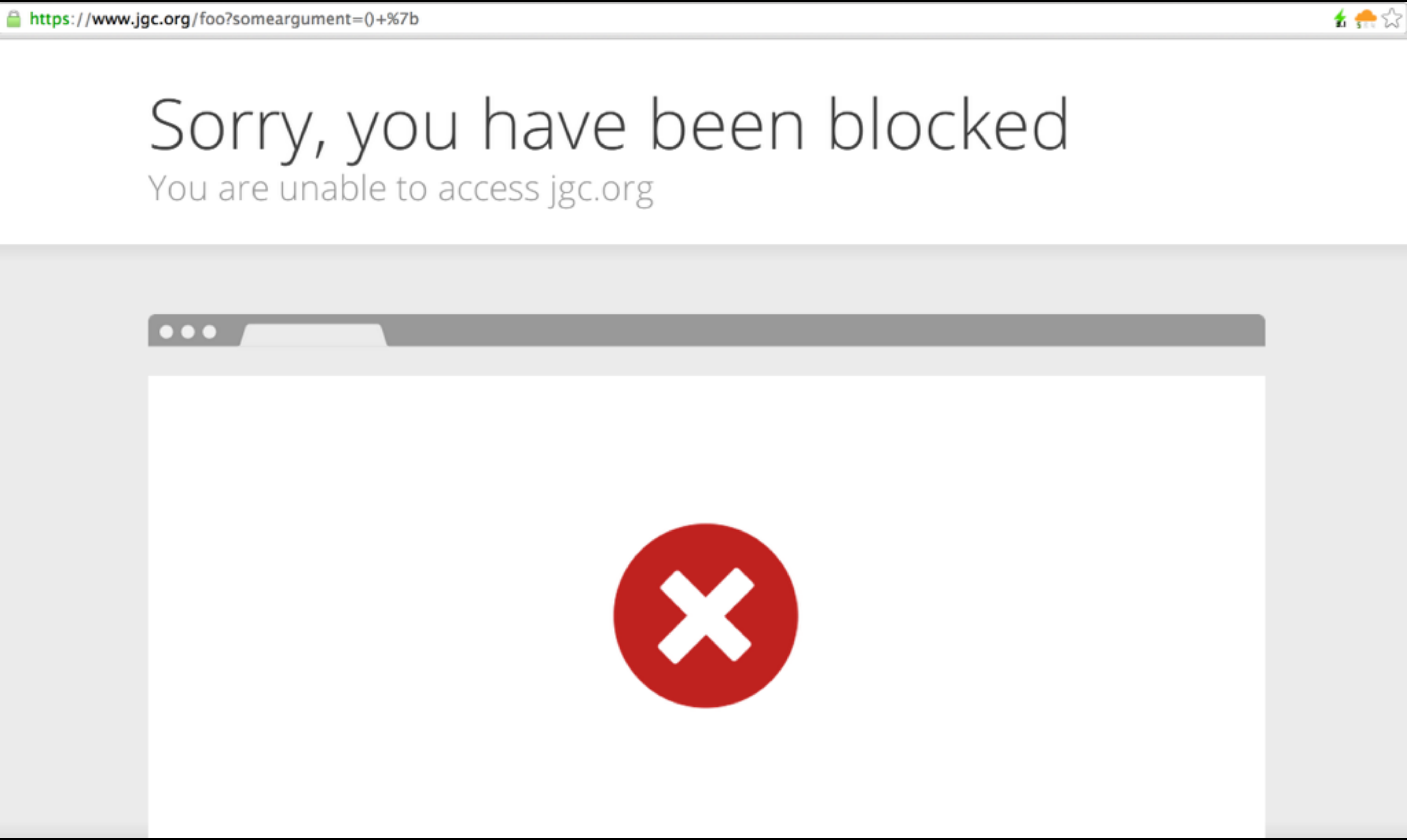
Two Things

ngx_lua/OpenResty Rules

Two Things

ngx_lua/OpenResty Rules

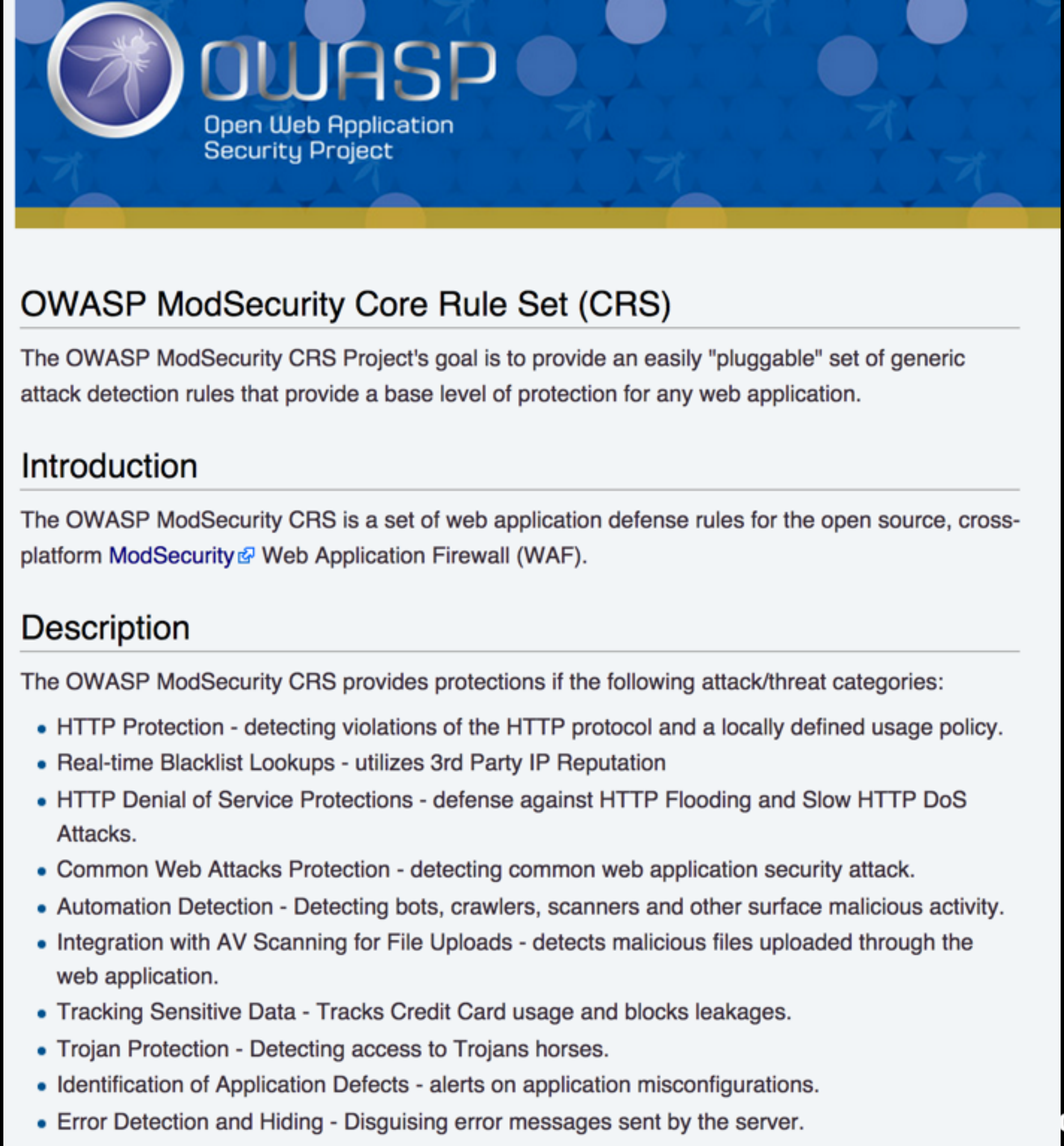
LuaJIT Rules



https://www.jgc.org/foo?someargument=()+%7b

Sorry, you have been blocked

You are unable to access jgc.org



OWASP
Open Web Application Security Project

OWASP ModSecurity Core Rule Set (CRS)

The OWASP ModSecurity CRS Project's goal is to provide an easily "pluggable" set of generic attack detection rules that provide a base level of protection for any web application.

Introduction

The OWASP ModSecurity CRS is a set of web application defense rules for the open source, cross-platform ModSecurity Web Application Firewall (WAF).

Description

The OWASP ModSecurity CRS provides protections if the following attack/threat categories:

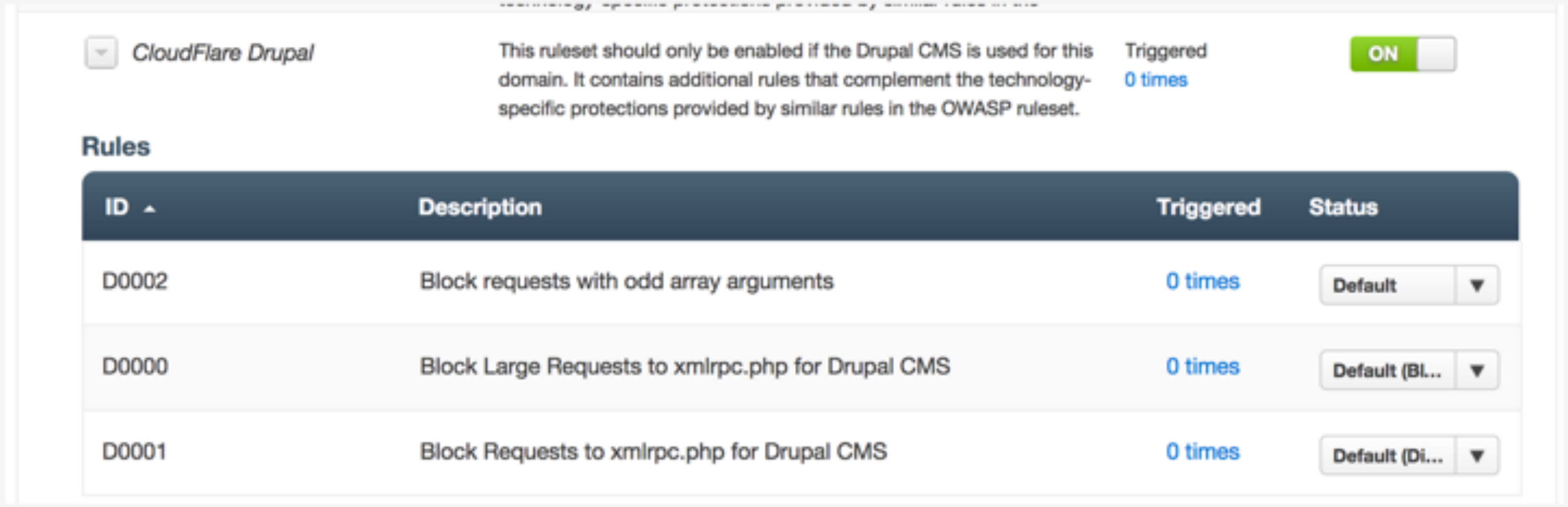
- HTTP Protection - detecting violations of the HTTP protocol and a locally defined usage policy.
- Real-time Blacklist Lookups - utilizes 3rd Party IP Reputation
- HTTP Denial of Service Protections - defense against HTTP Flooding and Slow HTTP DoS Attacks.
- Common Web Attacks Protection - detecting common web application security attack.
- Automation Detection - Detecting bots, crawlers, scanners and other surface malicious activity.
- Integration with AV Scanning for File Uploads - detects malicious files uploaded through the web application.
- Tracking Sensitive Data - Tracks Credit Card usage and blocks leakages.
- Trojan Protection - Detecting access to Trojans horses.
- Identification of Application Defects - alerts on application misconfigurations.
- Error Detection and Hiding - Disguising error messages sent by the server.

Drupal 7 SA-CORE-2014-005 SQL Injection Protection

16 Oct 2014 by John Graham-Cumming.

+1 9 Share 2 Like 55 Tweet 15

Yesterday the Drupal Security Team released a critical security patch for Drupal 7 that fixes a very serious SQL injection vulnerability. At the same time we pushed an update to our Drupal WAF rules to mitigate this problem. Any customer using the WAF and with the Drupal ruleset enabled will have received automatic protection.

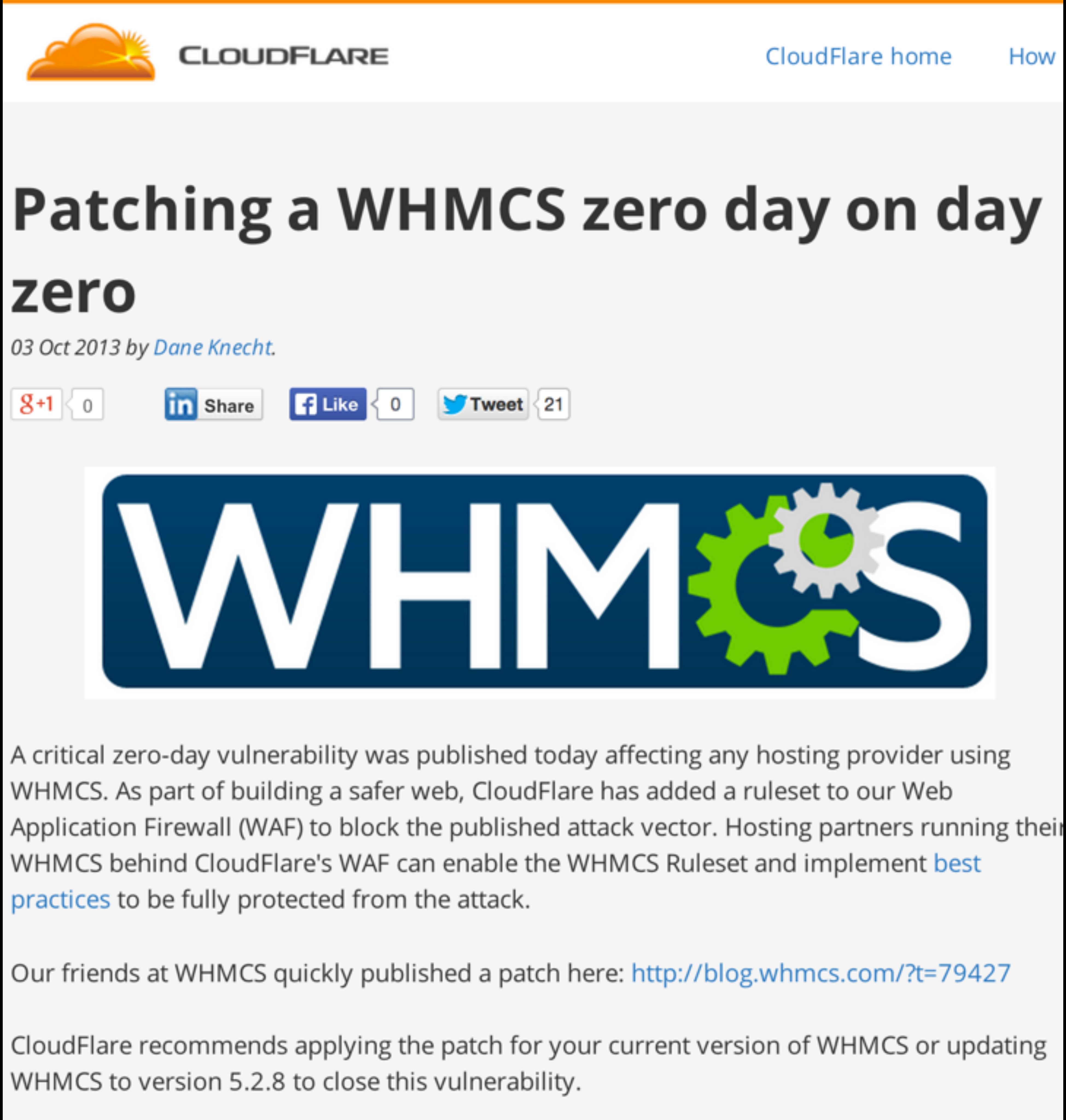


CloudFlare Drupal — This ruleset should only be enabled if the Drupal CMS is used for this domain. It contains additional rules that complement the technology-specific protections provided by similar rules in the OWASP ruleset. Triggered 0 times — ON

Rules

ID	Description	Triggered	Status
D0002	Block requests with odd array arguments	0 times	Default
D0000	Block Large Requests to xmlrpc.php for Drupal CMS	0 times	Default (Bl...
D0001	Block Requests to xmlrpc.php for Drupal CMS	0 times	Default (Di...

Rule D0002 provides protection against this vulnerability. If you do not have that ruleset enabled and are using Drupal clicking the ON button next to CloudFlare Drupal in the WAF Settings will enable protection immediately.



Patching a WHMCS zero day on day zero

03 Oct 2013 by Dane Knecht.

g+1 0 Share Like 0 Tweet 21

A critical zero-day vulnerability was published today affecting any hosting provider using WHMCS. As part of building a safer web, CloudFlare has added a ruleset to our Web Application Firewall (WAF) to block the published attack vector. Hosting partners running thei WHMCS behind CloudFlare's WAF can enable the WHMCS Ruleset and implement best practices to be fully protected from the attack.

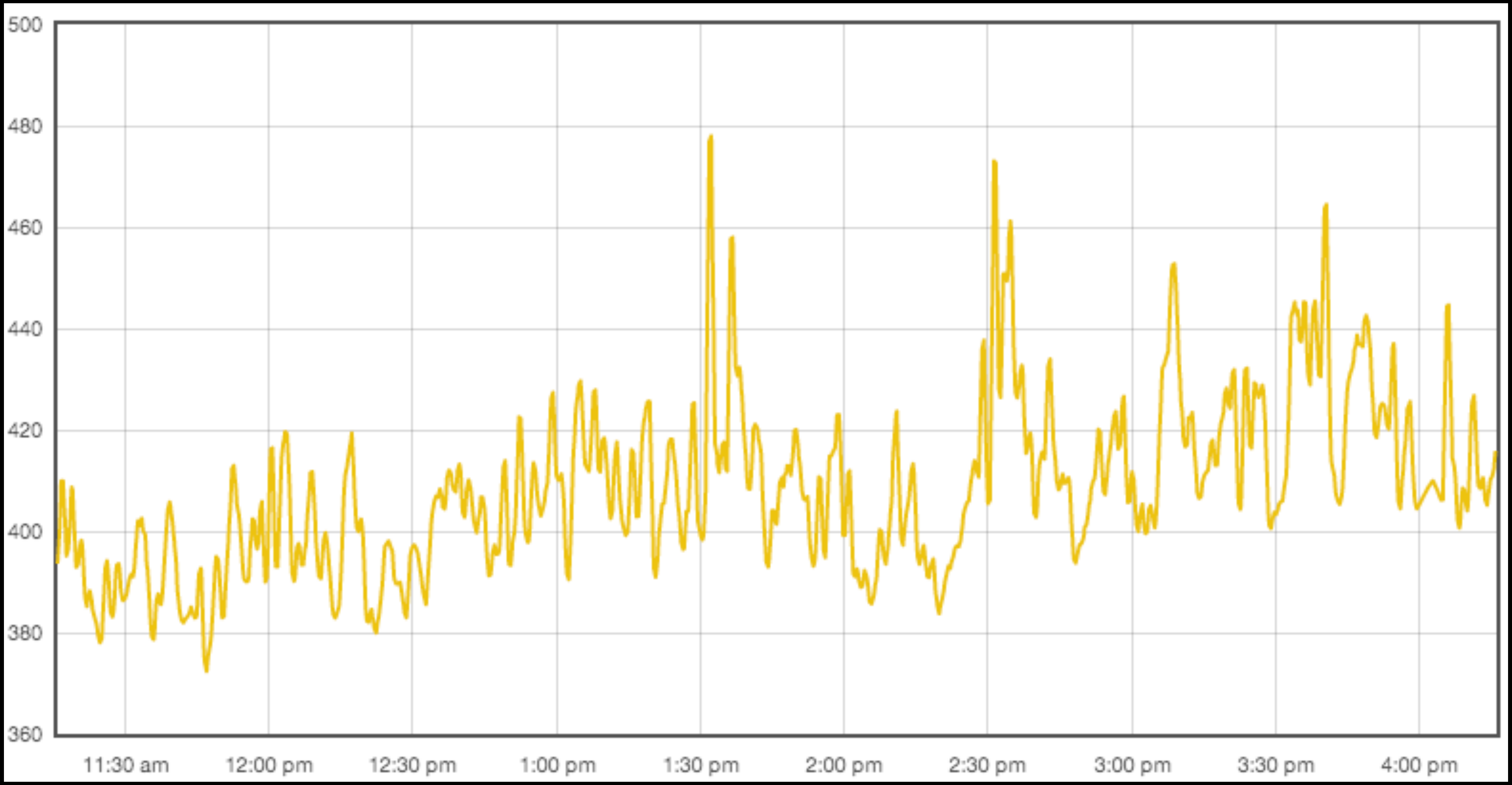
Our friends at WHMCS quickly published a patch here: http://blog.whmcs.com/?t=79427

CloudFlare recommends applying the patch for your current version of WHMCS or updating WHMCS to version 5.2.8 to close this vulnerability.

Rule counts

- 5,682 general rules plus 1,937 string matches
- 102 CloudFlare rules
- Customer specific rules
- BAD NEWS: worst case is all the time (run all rules)

<1ms latency

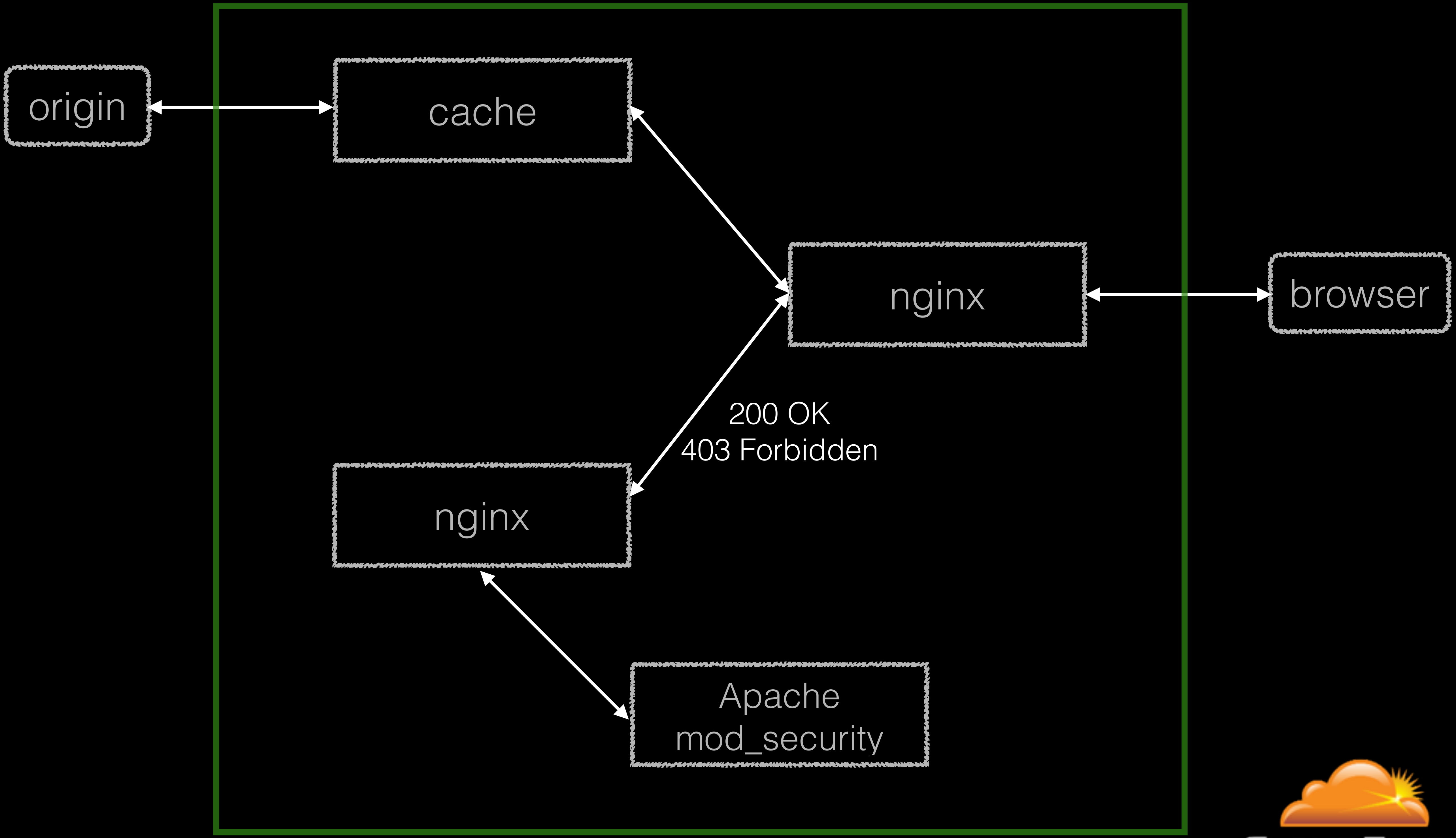


CLOUDFLARE

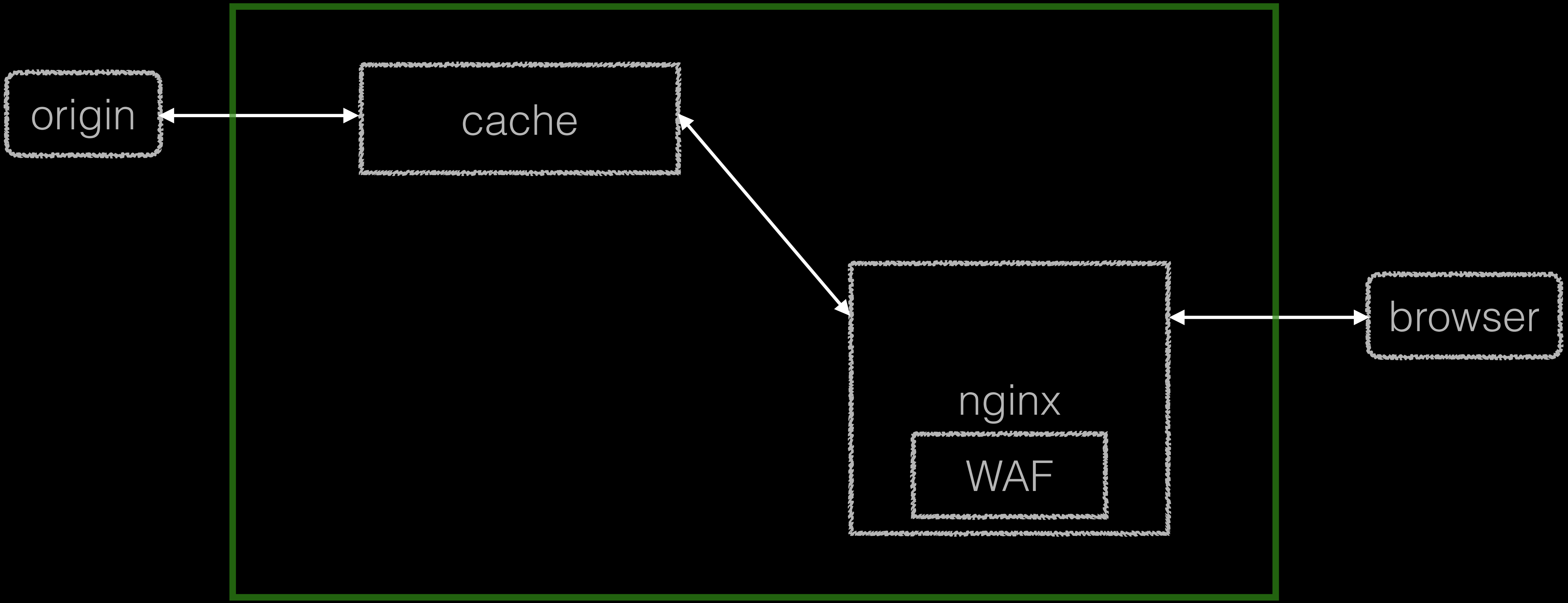
mod_security

- A well established Apache-based project to block bad HTTP requests (and responses)
- Didn’t scale for our needs (and we use nginx)
- I essentially rewrote it in Lua and it runs inside nginx
- agentzh did a lot of optimization work (*talk at 1145*)
- It’s very, very fast; it runs existing mod_security rules

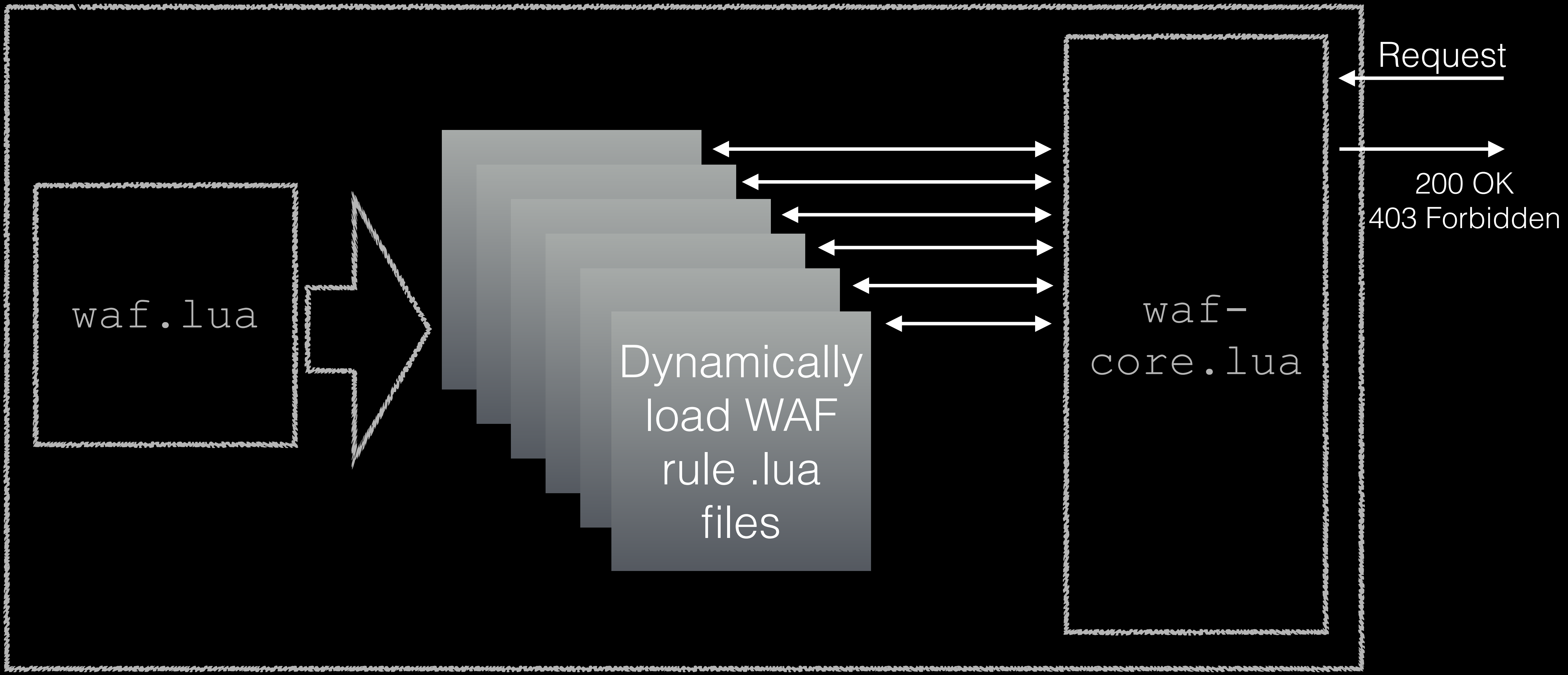
original architecture



new architecture



Inside WAF


Request
200 OK
403 Forbidden
waf.lua
Dynamically load WAF rule .lua files
waf-core.lua

nginx.conf

```
location / {
    set $backend_waf        "WAF_CORE";
    default_type            'text/plain';

    access_by_lua '
        local waf = require "waf"
        waf.execute("")
    ';

    log_by_lua_file "lua/metrics/waf_metrics_main.lua";

    content_by_lua 'ngx.say("")';
    error_page 500 =200 @error;
}
```

CLOUDFLARE

Lifecycle of a rule #1

Start with a mod_security rule like this:

```
SecRule ARGS:comment_post_id “@rx ^(\d+)$”
"drop,id:WP0005,msg:'Exploit DB 28485 Blind
SQL Injection’,phase:0"

SecRule VARIABLES FUNCTION ACTIONS/
TRANSFORMATIONS
```

Or our own language

```
SecRule REQUEST_HEADERS:User-Agent "@beginsWith DataStore/"
      "id:100000,phase:0,t:none,deny,chain,msg:'DataStore Attack'"
   SecRule REQUEST_METHOD "@streq GET" "chain"
      SecRule REQUEST_URI "\/\?-?\d+=-?\d+" ""
```

```
rule 100000 DataStore Attack
   REQUEST_HEADERS:User-Agent has-prefix DataStore/ and
   REQUEST_METHOD is GET and
   REQUEST_URI matches \/\?-?\d+=-?\d+
     deny
```

Lifecycle of a rule #2

It becomes a .json object

```
[{“source":"rulesets/cloudflare/enabled_rules/
cloudflare_01_wordpress.conf:60",
“actions":[  {"action":"drop"},
             {“parameter":"WP0005","action":"id"},
             {"parameter":"Exploit DB 28485 Blind SQL
Injection","action":"msg"},
             {“parameter":"0","action":"phase"}],
“default":"BLK",
“variables":["ARGS:comment_post_id"],
"description":"Exploit DB 28485 Blind SQL Injection”,
“operator":"^(/d+)$",
“overrides”:["DEF","DIS","SIM","BLK","CHL"],
“directive":"rule",
"id":"WP0005"}]
```

Lifecycle of a rule #3

And it turns into Lua code that an run in the WAF

```
 if not waf_disabled_ids['WP0005'] and waf_rx(waf, _M.v151_9, '151_9',
t151_1, '151_1', '^(/d+)$', false, nil, true) then
    waf_vars['RULE']['ID'] = 'WP0005'
    waf_activate(waf, _M.rulefile)
    waf_msg(waf, 'Exploit DB 28485 Blind SQL Injection')
    waf_drop(waf, _M.rulefile, false)
  end
```

Real example

```
local waf_vars = waf.vars
local waf_streq = waf.streq
local waf_setvar = waf.setvar
local waf_msg = waf.msg
local waf_drop = waf.drop
local waf_disabled_ids = waf.disabled_ids
local waf_deny = waf.deny
local waf_activate = waf.activate
local t1_1 = {}
if not waf_disabled_ids['00001'] and waf_streq(waf, v2_5, '2_5', t1_1, '1_1', 'b783efc191a7c066c1d87068f63a84a39f9830bb', false) then
  waf_vars['RULE']['ID'] = '00001'
  waf_activate(waf, rulefile)
  waf_msg(waf, 'CloudFlare Test Rule (drop) activated')
  waf_setvar(waf, {{'TX:ANOMALY_SCORE', '+100'},{'TX:%{RULE:ID}', 'CloudFlare unique hash test rule (drop)'}})
  waf_drop(waf, rulefile)
end
if not waf_disabled_ids['00002'] and waf_streq(waf, v2_5, '2_5', t1_1, '1_1', '4709edce126971876b547523778fa7b942ec14b5', false) then
  waf_vars['RULE']['ID'] = '00002'
  waf_activate(waf, rulefile)
  waf_msg(waf, 'CloudFlare Test Rule (deny) activated')
  waf_setvar(waf, {{'TX:ANOMALY_SCORE', '+100'},{'TX:%{RULE:ID}', 'CloudFlare unique hash test rule (deny)'}})
  waf_deny(waf, rulefile)
end
```

Available variables

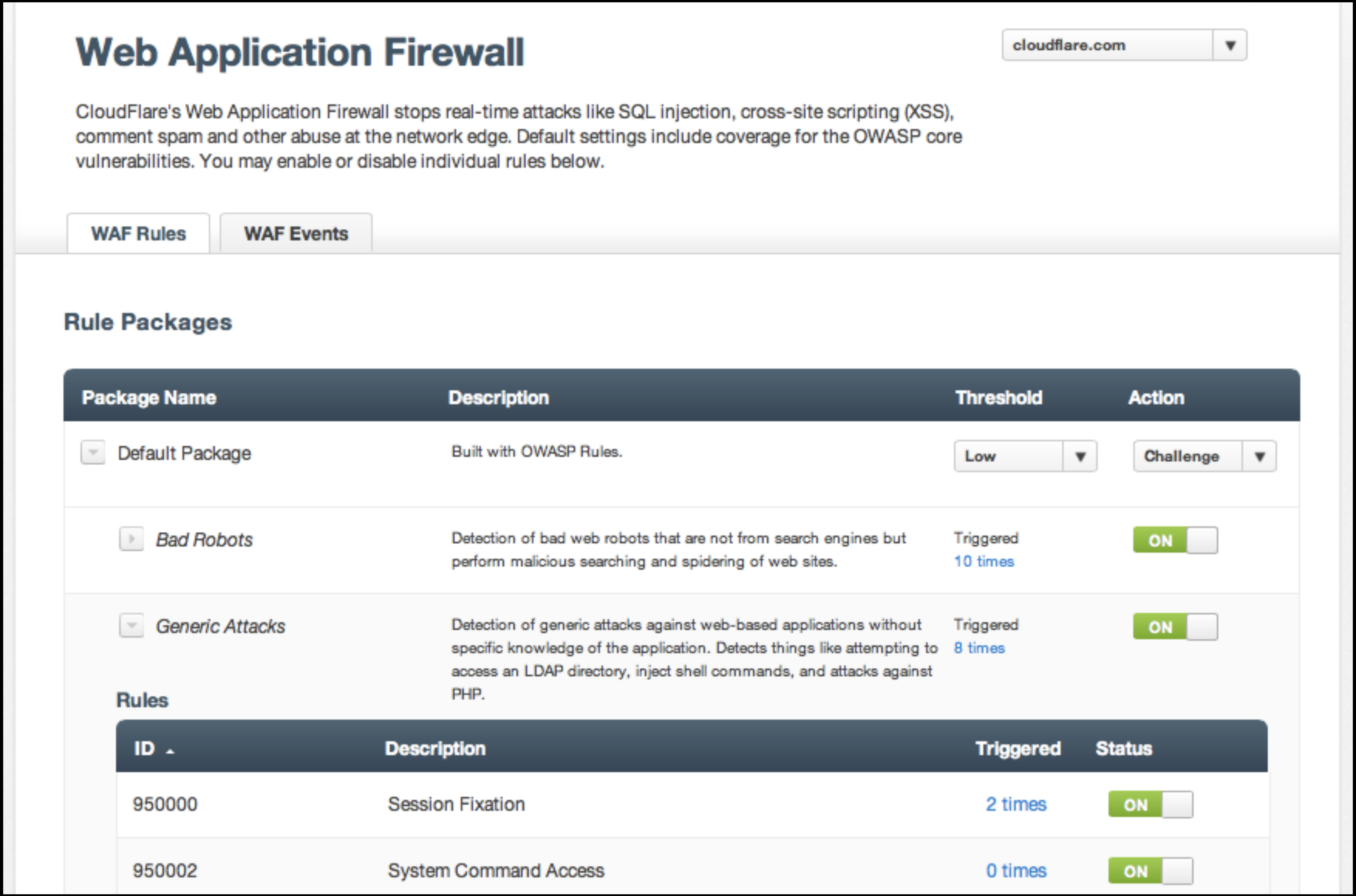
- Entire mod_security language works with our WAF
- `ARGS, ARGS_GET, ARGS_POST, ARGS_NAME,`
  `ARGS_COMBINED_SIZE`
  `AUTH_TYPE, QUERY_STRING, REMOTE_USER`
  `REQUEST_COOKIES, REQUEST_COOKIES_NAMES`
  `REQUEST_LINE, REQUEST_METHOD,`
  `REQUEST_PROTOCOL, REQUEST_URI`
  `REQUEST_HEADERS`
- `REQUEST_HEADERS:User-Agent, ARGS:user_id`

Matching functions

- All the mod_security matching functions
- Strings: **`@rx`** (PCRE regular expression), **`@contains`**, **`@containsWord`**, **`@beginsWith`**, **`@endsWith`**, **`@pm`** (match against multiple patterns), **`@streq`** (string equal), **`@empty`**, **`@nonEmpty`**, **`@hasNull`**, **`@within`**
- Numbers: **`@ge, @le, @gt, @lt, @eq, @ne`**
- Encodings: **`@validateUrlEncoding, @validateUtf8Encoding, @validateByteRange`**
- **`@verifyCC`** (check credit card number)

Transformations

- `lowercase, trim, trimLeft, trimRight, replaceComments, removeNulls, replaceNulls, removeWhitespace, compressWhitespace, hexEncode, hexDecode, base64Encode, base64Decode,`
- `urlDecode, urlEncode, cssDecode, jsDecode, escapeSeqDecode,`
- `md5, sha1, length`
- `normalisePath, normalisePathWindows,`



Web Application Firewall

cloudflare.com ▼

CloudFlare's Web Application Firewall stops real-time attacks like SQL injection, cross-site scripting (XSS), comment spam and other abuse at the network edge. Default settings include coverage for the OWASP core vulnerabilities. You may enable or disable individual rules below.

Rule Packages

Package Name	Description	Threshold	Action
Default Package	Built with OWASP Rules.	Low ▼	Challenge ▼
Bad Robots	Detection of bad web robots that are not from search engines but perform malicious searching and spidering of web sites.	Triggered 10 times	ON
Generic Attacks	Detection of generic attacks against web-based applications without specific knowledge of the application. Detects things like attempting to access an LDAP directory, inject shell commands, and attacks against PHP.	Triggered 8 times	ON

Rules

ID	Description	Triggered	Status
950000	Session Fixation	2 times	ON
950002	System Command Access	0 times	ON

Rule compiler optimizations

- Clause reordering so that rules can be quickly skipped if a sub-clause doesn't match
- Regular expression optimization and simplification
- Operator replacement so that fast operators (such as simple string matches) are used where possible
- Providing hints to the WAF runtime about whether macro expansion is needed.
- Global optimizations such as recognizing repeated use of the same strings or variables and ensuring that they are computed only once

Rule compiler optimizations

```
Swapping rx(\bactivexobject\b) for
containsWord(activexobject)

Swapping rx(,) for contains(,)

Swapping rx(^$) for empty

Swapping rx(^(.*)$) for unconditionalMatch

Swapping rx(^GET /$) for streq(GET /)

Swapping rx(Mozilla\/5\.0 \(compatible; MSIE 9\.0; Windows
NT 6\.1; WOW64; Trident\/5\.0; SLCC2; Media Center PC 6\.
0; InfoPath\.3; MS-RTC LM 8; Zune 4\.7\)) for
contains(Mozilla/5.0 (compatible; MSIE 9.0; Windows NT
6.1; WOW64; Trident/5.0; SLCC2; Media Center PC 6.0;
InfoPath.3; MS-RTC LM 8; Zune 4.7))
```

CLOUDFLARE

Other Optimization

- Basic Lua optimizations
- Deep dive into regular expressions
- Special array handling
- Make sure code is JITable
- Caching
- Memoization

Basic Optimization

Wait til you've finished; Measure; Fix the slow things

Locals way faster than globals

```
local rand = math.random

local len = #t
for i=1,len do
    ...
end
```

`.` syntax faster than `:`

```
local slen = string.len
s:len() vs. slen(s)
```

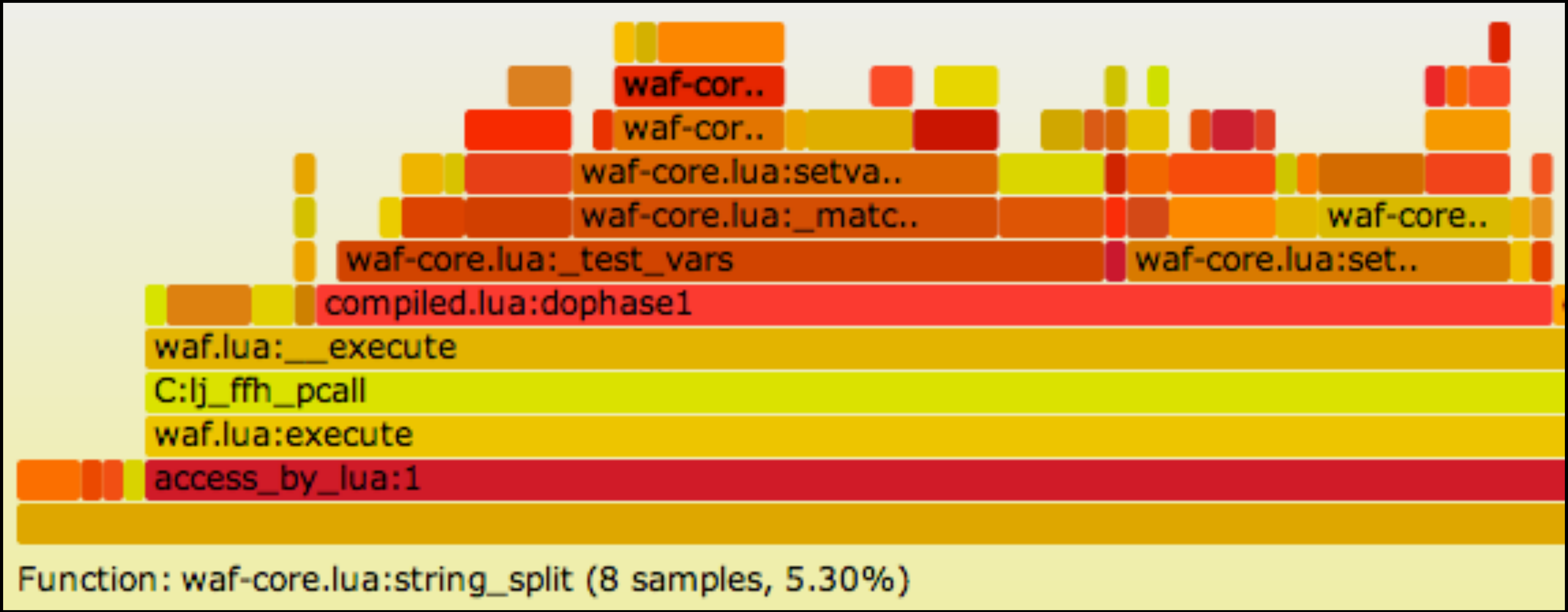
Minimize closures

Basic Optimization

Use PCRE JIT and caching options

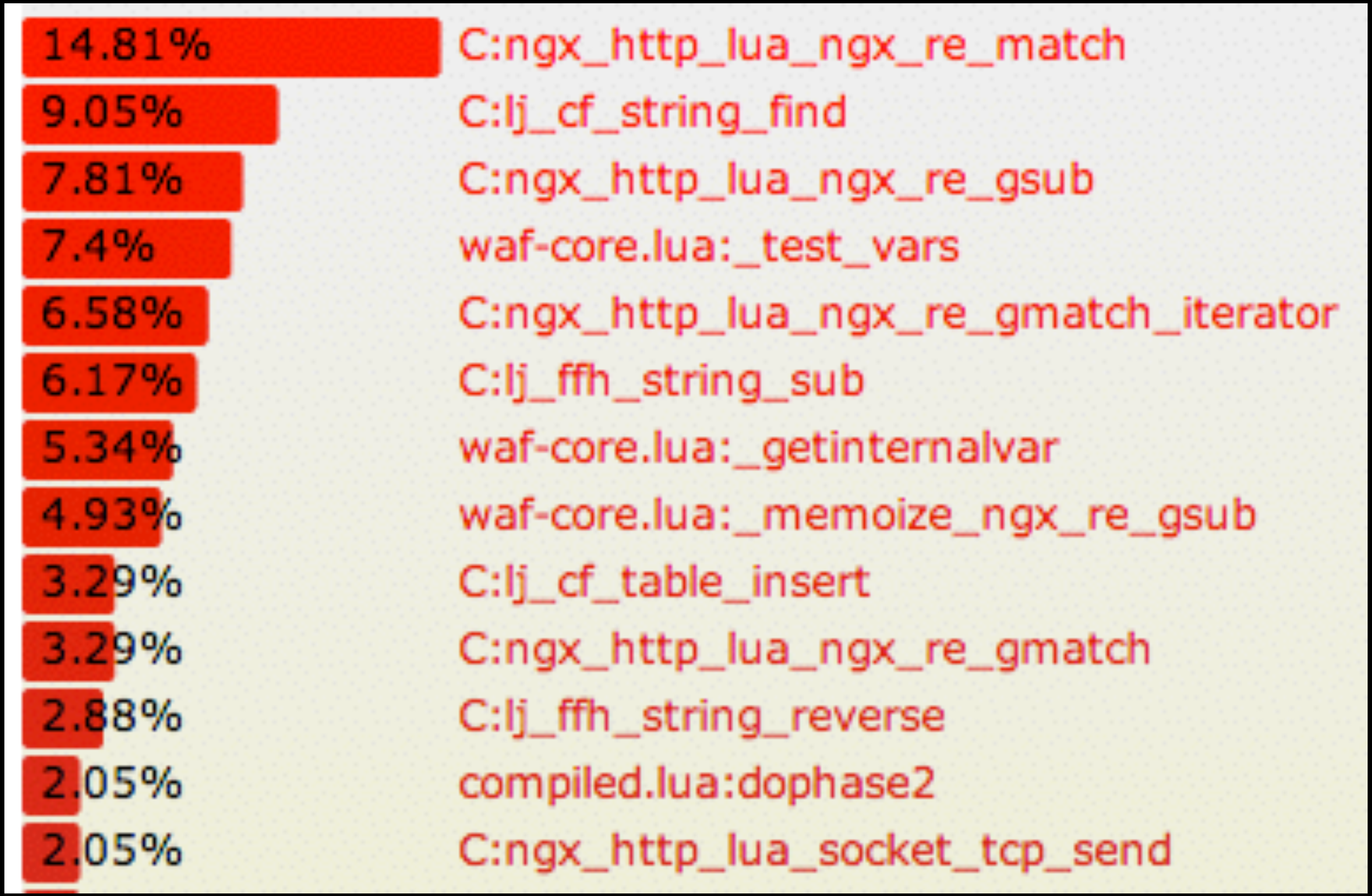
Use `lua_shared_dict` to cache frequently accessed items

Lua Flamegraph



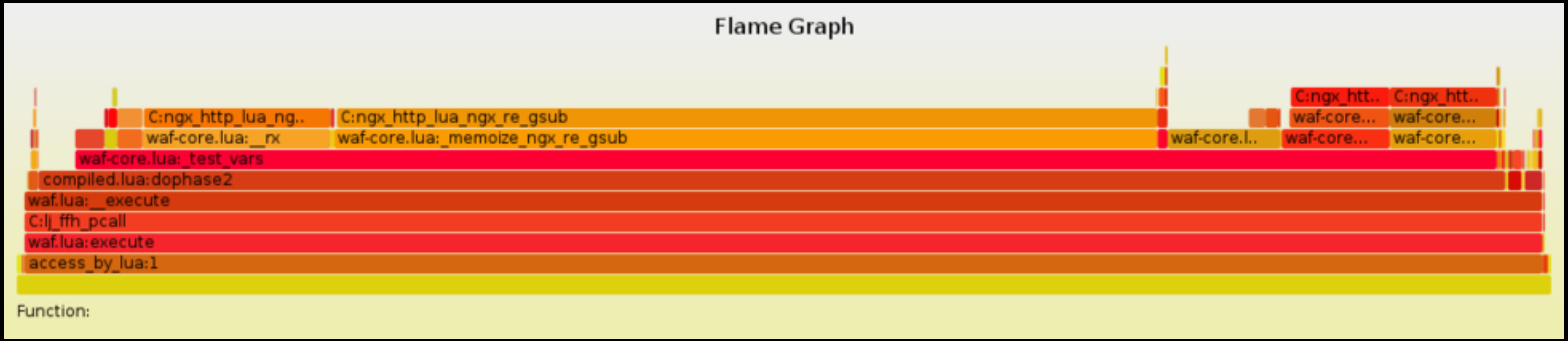
Function: waf-core.lua:string_split (8 samples, 5.30%)

Top functions



14.81%	C:ngx_http_lua_ngx_re_match
9.05%	C:lj_cf_string_find
7.81%	C:ngx_http_lua_ngx_re_gsub
7.4%	waf-core.lua:_test_vars
6.58%	C:ngx_http_lua_ngx_re_gmatch_iterator
6.17%	C:lj_ffh_string_sub
5.34%	waf-core.lua:_getinternalvar
4.93%	waf-core.lua:_memoize_ngx_re_gsub
3.29%	C:lj_cf_table_insert
3.29%	C:ngx_http_lua_ngx_re_gmatch
2.88%	C:lj_ffh_string_reverse
2.05%	compiled.lua:dophase2
2.05%	C:ngx_http_lua_socket_tcp_send

C Flamegraph



PCRE monitoring

```
$ ./ngx-pcre-stats -p 24528 --exec-time-dist
 Tracing 24528 (/path/to/nginx/sbin/nginx)...
 Hit Ctrl-C to end.
 ^C
 Logarithmic histogram for pcre_exec running time distribution
(us):
 value |-------------------------------------------------- count
     0 |                                                       0
     1 |                                                       0
     2 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                     981
     4 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  1479
     8 |                                                      16
    16 |                                                      18
    32 |                                                       1
```

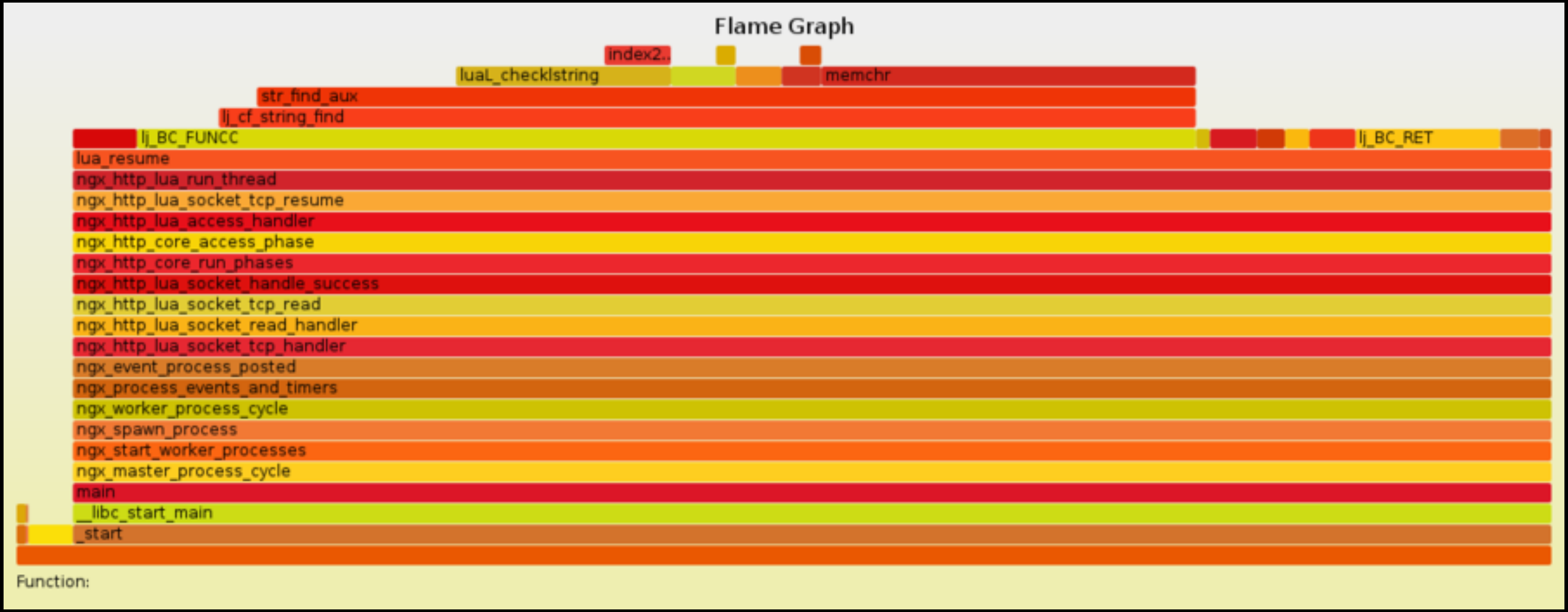
Regexp dive

```
$ ./ngx-pcre-stats -p 24528 --total-time-top --luajit20
 Tracing 24528 (/path/to/nginx/sbin/nginx)...
 Hit Ctrl-C to end.
 ^C
 Top N regexes with longest total running time:
 1. pattern /WEB_ATTACK/: 15103us (total data size: 82184)
 2. pattern / __cf__\d+/: 11143us (total data size: 25916)
 3. pattern /[^\x01-\xff]/: 10233us (total data size:
102825)
 4. pattern /\b(?:coalesce\b|root\@)/: 7017us (total data
size: 78230)
 5. pattern /(Content-Length|Transfer-Encoding)/: 6766us
(total data size: 17871)
 ...
```

Slow regexps

```
$ ./ngx-pcre-stats -p 24528 --worst-time-top --luajit20
 Tracing 24528 (/path/to/nginx/sbin/nginx)...
 Hit Ctrl-C to end.
 ^C
 Top N regexes with worst running time:
 1. pattern /\.cookie\b.*?\;\W*?domain\W*?\=/: 98us (data size: 36)
 2. pattern /(Content-Length|Transfer-Encoding)/: 89us (data size: 
14)
 3. pattern / __cf__\d+/: 63us (data size: 8)
 4. pattern /[^\x01-\xff]/: 53us (data size: 13)
 5. pattern /\b(background|dynsrc|href|lowsrc|src)\b\W*?=/: 53us 
(data size: 5147)
 6. pattern /(?i:<embed[ /+\t].*?SRC.*?=)/: 47us (data size: 304)
 7. pattern /(fromcharcode|alert|eval)\s*\(/: 45us (data size: 24)
 8. pattern /\bselect\b.*?\bto_number\b/: 40us (data size: 5147)
```

Something's spinning



Peek inside backtrace

```
$ ./ngx-lua-bt -p 7599 --luajit20
 WARNING: Tracing 7599 (/path/to/nginx/sbin/nginx) for
LuaJIT 2.0...
 C:lj_cf_string_find
 @/waf/lua/waf-core.lua:201
 @/waf/lua/waf-core.lua:676
 @/waf/lua/waf-core.lua:1467
 @/waf/lua/waf-core.lua:1074
 @/waf/lua/rules/oldwaf/compiled.lua:371
 @/waf/lua/waf.lua:57
 C:lj_ffh_pcall
 @/waf/lua/waf.lua:50
 access_by_lua:1
```

Tools

https://github.com/openresty/nginx-systemtap-toolkit/

https://github.com/brendangregg/FlameGraph

Two step caching

- Compiled Lua stored in memcached
- Once loaded stored in `lua_shared_dict`

Cached `require`

```
if not package.loaded[r] then
   local src, flags, err = memc:get(filekey)
   local m, err = loadstring(src, filekey)

   if m then
      package.loaded[r] = m()
  end
end

local loaded = require(r)
if not loaded then
   package.loaded[r] = nil
   return nil, "require(" .. r .. ") returned nil"
else
   return loaded, nil
end
```

LuaJIT

- Is a *tracing* compiler for Lua
- Fast!
- Key is to make sure it can JIT
- Has special APIs
- Sponsor Mike Pall!

LuaJIT `table` API

-
  ```
  tn = require("table.new")
  t = tn(narr, nrec)
  ```

Dual use arrays

- Store both array style and hash style in same array
- Get fast iteration of keys or values

```
days[0] = 0
days[1] = "January"
days[2] = 31
days[days[1]] = days[2]
days[0] = days[0] + 2
…

for i = 1, days[0], 2 do
   k = days[i]
   v = days[i+1]
end
```

```
function add(t, m, d)
    t[m] = d
    t[t[0]+1] = m
    t[t[0]+2] = d
    t[0] = t[0] + 2
end

local tn = require("table.new")
local t = tn(10, 10)
t[0] = 0

add(t, "January", 31)
add(t, "February", 28)
add(t, "March", 31)
add(t, "April", 30)
add(t, "May", 31)
add(t, "June", 30)
add(t, "July", 31)
add(t, "August", 31)
add(t, "September", 30)
add(t, "October", 31)
add(t, "November", 31)
add(t, "December", 31)

for i = 1, t[0], 2 do
    print(t[i], " has ", t[i+1], " days")
end

print("October has ", t["October"], " days")
```

```
% luajit -bl test.lua
-- BYTECODE -- test.lua:1-6
0001    TSETV     2   0   1
0002    TGETB     3   0   0
0003    ADDVN     3   3   0  ; 1
0004    TSETV     1   0   3
0005    TGETB     3   0   0
0006    ADDVN     3   3   1  ; 2
0007    TSETV     2   0   3
0008    TGETB     3   0   0
0009    ADDVN     3   3   1  ; 2
0010    TSETB     3   0   0
0011    RET0      0   1

-- BYTECODE -- test.lua:0-31
0001    FNEW      0   0      ; test.lua:1
0002    GSET      0   1      ; "add"
0003    GGET      0   2      ; "require"
0004    KSTR      1   3      ; "table.new"
0005    CALL      0   2   2
0006    MOV       1   0
0007    KSHORT    2  10
0008    KSHORT    3  10
0009    CALL      1   2   3
0010    KSHORT    2   0
0011    TSETB     2   1   0
0012    GGET      2   1      ; "add"
0013    MOV       3   1
0014    KSTR      4   4      ; "January"
0015    KSHORT    5  31
0016    CALL      2   1   4
0017    GGET      2   1      ; "add"
0018    MOV       3   1
0019    KSTR      4   5      ; "February"
0020    KSHORT    5  28
0021    CALL      2   1   4
0022    GGET      2   1      ; "add"
0023    MOV       3   1
0024    KSTR      4   6      ; "March"
0025    KSHORT    5  31
0026    CALL      2   1   4
```

LuaJIT 2.1 now JITable

- CAT (concatenate operator)
- FUNC, FUNCC (C function calls)
- TSETM (return table with multiple values)
- string.char, string.find, string.format, string.lower, string.rep, string.reverse, string.upper
- table.concat, table.foreachi, table.remove

LuaJIT FFI

```
local ffi = require 'ffi'
local C = ffi.C

ffi.cdef[[
  typedef long time_t;

  typedef struct timeval {
    time_t tv_sec;
    time_t tv_usec;
  } timeval;

  int gettimeofday(struct timeval* t, void* tzp);
]]

local gettimeofday_struct = ffi.new("timeval")

-- _gettimeofday_: wrapper function that calls the C gettimeofday
-- function via FFI and returns a value in microseconds
local function gettimeofday()
   C.gettimeofday(gettimeofday_struct, nil)
   return tonumber(gettimeofday_struct.tv_sec) * 1000000 +
      tonumber(gettimeofday_struct.tv_usec)
end
```

Closing Thoughts

ngx_lua/OpenResty Rules

LuaJIT Rules

PCRE Rules