A Little WAF

detailyang@gmail.com

2016.10

A Web Application Firewall (or WAF) is a firewall that filters, monitors, and blocks HTTP/S traffic to and from a web application

Background

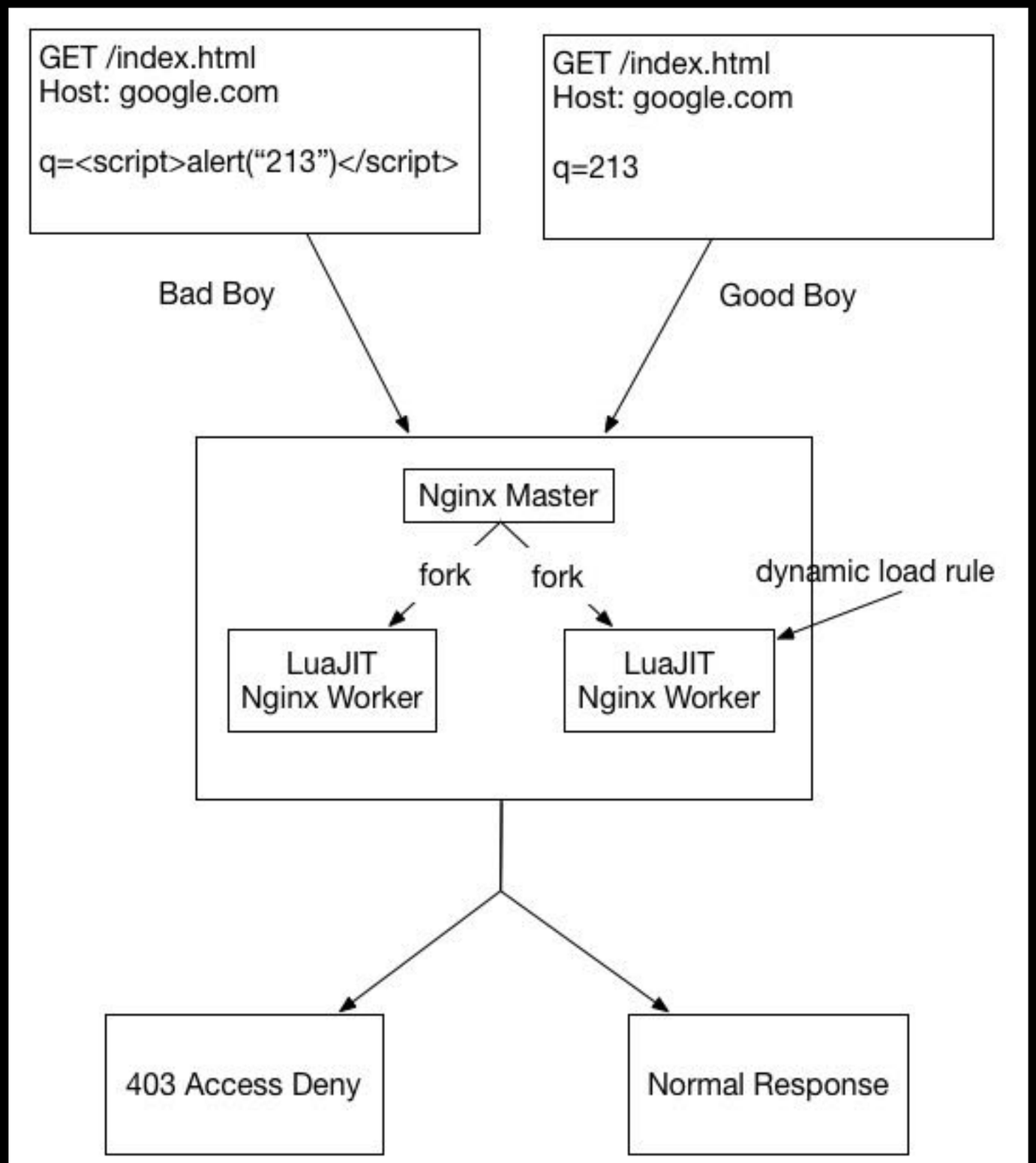
1. IP Deny — Blacklist and Whitelist with time expire
2. Common Web Attacks Protection like XSS、SQLInject

Principe

According the cloudflare blog post in 2013 cloudflares-new-waf-compiling-to-lua

https://blog.cloudflare.com/cloudflares-new-waf-compiling-to-lua/

Architecture



Nginx Conf

```
http {
    init_by_lua_block {
        local wafinit = require("bkb.waf-init")
        wafinit.run("/tmp/waf")
    }
    access_by_lua_block {
        local waf = require("bkb.waf")
        waf.run()
    }
    log_by_lua_block {
        local waflog = require("bkb.waf-log")
        waflog.run()
    }
}
```

ModSecurity

Open Source Web Application Firewall

https://modsecurity.org/

apache module ModSecurity

Now Nginx Plus Support ModSecurity

OWASP ModSecurity Core Rule Set (CRS)

https://github.com/SpiderLabs/owasp-modsecurity-crs

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

```
SecRule URI “@endsWith .git”
“drop,id:WP0006,msg:'Scan .GIT
Directory’,phase:0”
```

Key Concept

```
if operator(transform(variable), pattern) then
  action
else
  continue
end
```

Variable

POST / HTTP/1.1 (Request Line)
Host: www.google.com (Request Header)
Connection: keep-alive
Content-Length: 5
Cookie: a=1;b=2;

q=123 (Request Body)

Available Variables

'ip', 'uri', 'request_headers', 'request_cookies', 'args', 'matched_var'

Available Operators

'eq', 'rx', 'ipMatch', 'beginsWith', 'endsWith', 'ge', 'gt', 'lt', 'le', 'empty', 'nonEmpty', 'within', 'pmFromFile', 'pm'

Available Transforms

'urlDecodeUni', 'jsDecode', 'lowercase', 'base64Decode',
'base64Encode',
'length', 'sha1', 'htmlEntityDecode', 'compressWhitespace',
'removeWhitespace',
'cssDecode'

Available Actions

'deny', 'skip', 'log'

WAF Rule Format

```
{
    "id": 1,
    "phase": "access",
    "scope": "uri",
    "tag": 4,
    "chain": [],
    "operator": {"name": "rx", "reverse": false},
    "variable": [
        {
            "key": "",
            "match": "eq",
            "name": "uri",
            "reverse": false
        }
    ],
    "transform": [],
    "pattern": {"type": "string", "value": "\\.(git|svn)$"},
    "action": [
        {"name": "deny", "param": "null"}
    ]
}
```

Rule To Lua

```
if waf_operator_rx(waf, ctx, '1',  false ,••
        t._0, 't._0', v._0, 'v_0', [==[\.(git|svn)$]==]) then
    waf_action_deny(waf, ctx, '1', [==[null]==])
    return
end
```

Rule To Test Code

```
7  our $http_config = <<"_EOC_";
8      lua_shared_dict wafrule 10m;
9      lua_package_path '$workdir/?.lua;;';
10     lua_package_cpath '$workdir/bkb/clib/?.so;;';
11     access_by_lua '
12         local waf = require "bkb.waf"
13         waf.use_x_forwarded_for = true
14         waf.run()
15     ';
16 _EOC_
17
18 repeat_each(1);
19 no_shuffle();
20 run_tests();
21
```

Rule To Test Code

```
__DATA__

=== TEST1: test 1

--- http_config eval: $::http_config

--- config
location ~ .* {
    content_by_lua 'ngx.say("hello world")';
}

--- raw_request eval
"
GET /.git HTTP/1.0
Host: localhost

"

--- error_code: 403
```

Hot Load Rules

base on Lua global table

package.loaded

loadstring

```
local chunk, err = loadstring(code, ‘=rule.lua’)

if not err then
  package.loaded[‘rule’] = chunk()
end
```

PS: LuaVM

Dry Mode and Run Mode

dry mode: only logging
run mode: dis/enable WAF

Dry Mode and Run Mode

lua_shared_dict

Dry Mode and Run Mode

base on lua_shared_dict

multi process are communicated with shared memory

Side Effect

every request will try to get four locks:
ip lock、rule lock、dry lock 、run lock.

ngx_shmtx_lock(&ctx->shpool->mutex);

Side Effect

every request will try to get four locks:
ip lock、rule lock、dry lock 、run lock.

ngx_shmtx_lock(&ctx->shpool->mutex);

lock time O(log n) based on red black tree

Logging

lua-resty-logger-socket

base on cosocket

Logging

RFC 5424 to rsyslog server

delay

max: 20ms min: 5us avg: 350us

API For OP

```
server {
  listen 80;
  server_name bkb;

  allow 127.0.0.1;
  allow 10.10.0.0/16;
  deny all;

  location / {
    content_by_lua_block {
      local wafapi = require("bkb.waf-api");
      wafapi.run("/data/waf/mode")
    }
  }
}
```

API For OP

curl -H "Host: bkb” "http://$hostname/waf"

waf dashboard for monitor

PS: tsar support

API For OP

curl -H "Host: bkb” "http://$hostname/waf"


```
{
    totaldelay: 2263817,
    waf_mode_file: "/tmp/waf",
    rule: { version: xxxx},
    ip: { version: yyyyy},
    delay: 144.884288,
    dry: false,
    totalcnt: 15625,
    trigger: 2,
    run: true
}
```

API For OP

curl -H "Host: bkb" -X POST -d "dry=1" "http://$hostname/waf"

let waf enter dry mode with fs persistence

curl -H "Host: bkb" -X POST -d "dry=1" "http://$hostname/waf"

let waf exit dry mode

API For OP

curl -H "Host: bkb" -X POST -d "run=1" "http://$hostname/waf

let waf enable run mode with fs persistence

curl -H "Host: bkb" -X POST -d "run=0" "http://$hostname/waf

let waf disable run mode

API For OP

curl -H "Host: bkb" -X PUT "http://$hostname/rule"

hot load the new rule

curl -H "Host: bkb" -X PUT "http://$hostname/ip"

hot load the new ip

Any questions ?