# Apache Traffic Server & Lua

Kit Chan (kichan@yahoo-inc.com)

APACHECON EUROPE

CORINTHIA HOTEL
BUDAPEST, HUNGARY
NOVEMBER 17-21, 2014

# Agenda

- Intro
- Rationale
- Details
- Security
- Future

# Apache Traffic Server

Fast, scalable and extensible HTTP/1.1 compliant caching proxy server

# History with Yahoo

# Lua

Fast, Powerful, lightweight, embeddable scripting language

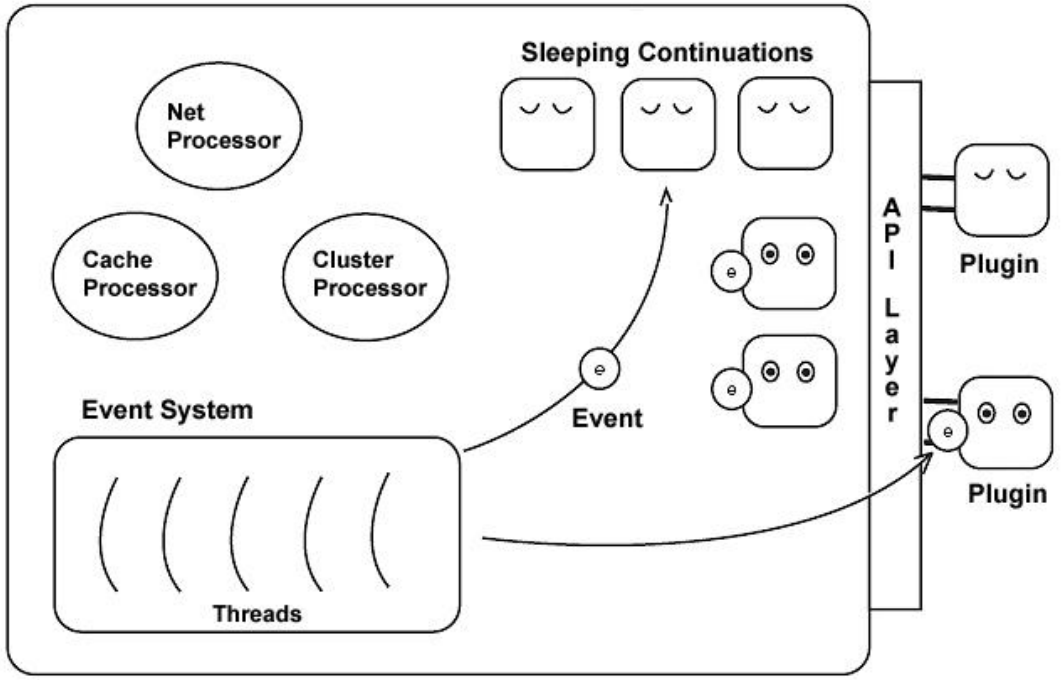
# Lua Uses


- mod_lua
- ngx_lua
- Redis
- MySQL Proxy
- some games/game engines(e.g World of Warcraft)

# What Can They Do Together?

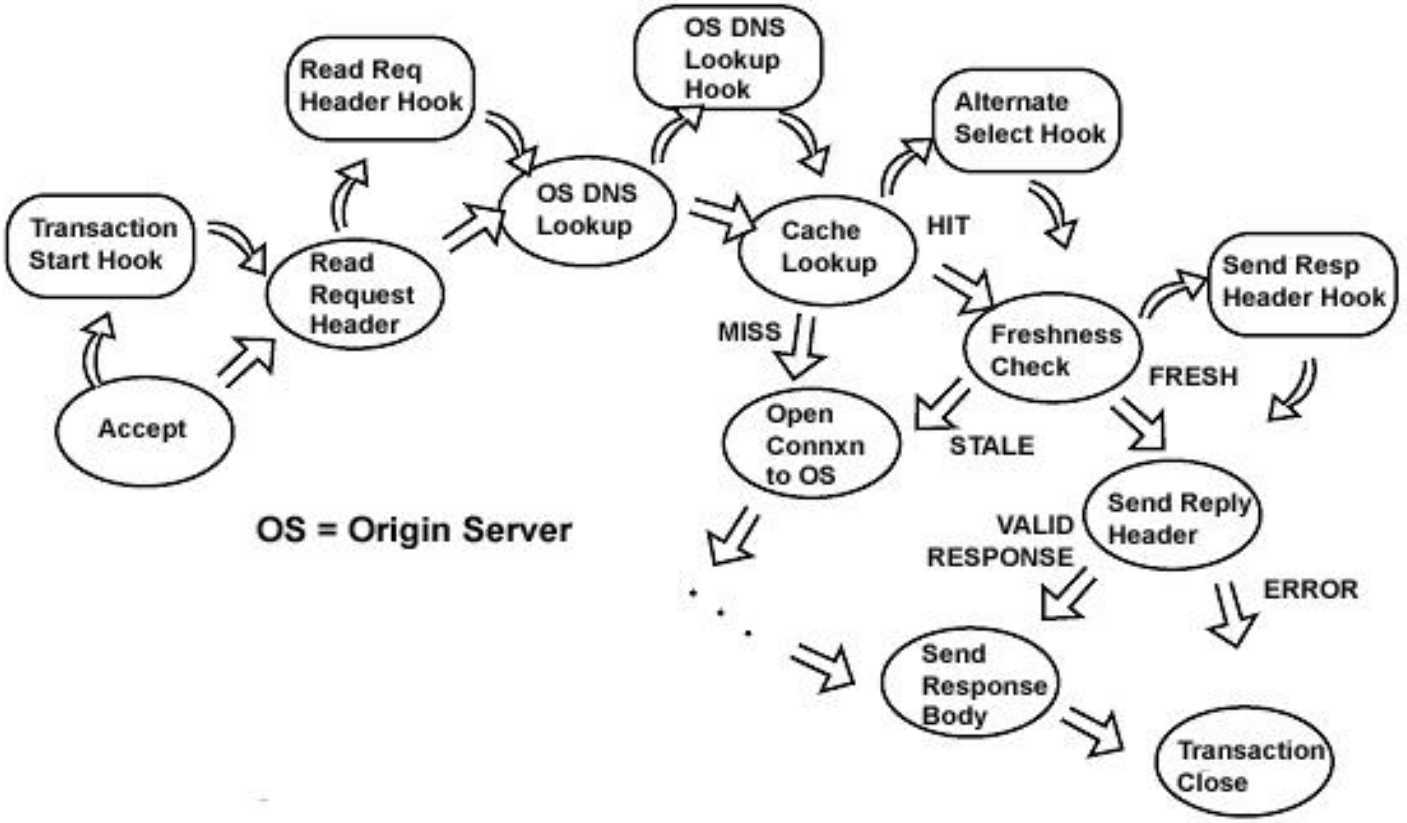
# Apache Traffic Server API

- C APIs
  - Http Header
  - Http Transaction
  - Statistics
  - Management
  - More!!!



# Hooks & Plugins

- Plugins implemented as Continuations
- Hooks to various HTTP transaction state



# Writing Plugins is Hard!

- Continuation is hard to follow
  - “Gotos with arguments”
- C/C++ code
  - Memory management
  - Complex data structure

# header_rewrite plugin

- No C code for simple tasks
- Use configuration files

# header_rewrite plugin

```
# sample header_rewrite configuration file
# add Cache-Control header
cond %{READ_RESPONSE_HEADER_HOOK} [AND]
rm-header Cache-Control
add-header Cache-Control “max-age=0, public” [L]

# primitive boolean operator
cond %{READ_REQUEST_HEADER_HOOK} [AND]
cond %{HEADER:Host} “news.yahoo.com” [OR]
cond %{HEADER:Host} “sports.yahoo.com”
rm-header X-Important
add-header X-Important 1 [L]
```

# header_rewrite plugin

- Hard to use
  - Cannot define variables
  - No loop
  - No module/function
- No unit test framework

# Varnish

- Major Competitor to ATS

# Varnish Configuration Language (VCL)

- DSL to configure Varnish
- Things you can do
  - “direct certain requests to certain backends”
  - “alter the requests and the responses”
- Major advantage over ATS !!!

# VCL

```
//sample VCL to
//remove cookie from image reqs
sub vcl_recv {
  if (req.url ~ "^/images") {
    unset req.http.cookie;
  }
}

// another example to cache response for 2 minutes
// if there is no cache control header
sub vcl_fetch {
  if (beresp.ttl < 120s) {
    set beresp.ttl = 120s;
  }
}
```

# Leveling the field using Lua


- Scripting Language
  - Simple to learn and use
  - Abstract user from variable types
  - No worry to memory management
- Extensible through C APIs
- Easy to embed into applications, such as ATS

# All the Glory Details

# Implementation Details

- ATS C APIs are wrapped as Lua Functions under a ‘ts’ module
- When ATS starts,
  - Creates and inits Lua VM (lua_newstate())
  - Load a Lua script into VM
  - Register hooks with functions in script
- During a HTTP transaction handled by ATS,
  - New Lua thread (lua_newthread()) created for VM
  - Call Lua functions registered to HTTP transaction hooks

# Sample

```
-- Lua script (cors.lua) to add CORS header to response
-- for a request with matching Referer header
function send_response()
    if ts.ctx['origin'] == nil then
      ts.debug("invalid referer");
    else
      ts.client_response.header['Access-Control-Allow-Origin']=ts.ctx['origin']
    end
    return 0
end

function do_global_read_request()
    local referer = ts.client_request.header.Referer
    if referer == nil then
      ts.ctx['origin'] = nil
    else
      ts.ctx['origin'] = string.match(referer, "http://%a+.yahoo.com")
    end
    ts.hook(TS_LUA_HOOK_SEND_RESPONSE_HDR, send_response)
    return 0
end
```

# Sample

- We refer to this script file in plugin.config

```
# inside /usr/local/etc/trafficserver/plugin.config

tslua.so /usr/local/etc/trafficserver/cors.lua
```

# Using Module

```
-- cors_lib.lua contains module for adding CORS resp header
-- for a request with matching Referer header
local cors_lib = {}
function cors_lib.send_response()
  ts.client_response.header['Access-Control-Allow-Origin'] = ts.ctx['origin‘]
  return 0
end

function cors_lib.execute()
  local referer = ts.client_request.header.Referer
  if referer ~= nil then
    ts.ctx['origin'] = string.match(referer, "http://%a+.yahoo.com")
    if ts.ctx['origin'] ~= nil then
      ts.hook(TS_LUA_HOOK_SEND_RESPONSE_HDR, send_response)
    end
  end
end
return cors_lib
```

# Using Module

```
-- main lua script (main.lua) to be refererd in plugin.config

-- read in the module defined in separate files
ts.add_package_path('/usr/local/etc/trafficserver/cors_lib.lua')

local cors_lib = require “cors_lib"

function do_global_read_request()
  cors_lib.execute()
  return 0
end
```

# Using Module

- We can refer to the main script file in plugin.config

```
# inside /usr/local/etc/trafficserver/plugin.config

tslua.so /usr/local/etc/trafficserver/main.lua
```

# Other Details

- Open Source
  - Contributers from Yahoo and Taobao
- Detailed Documentation Available
- ATS 5.x
  - Luajit 2.0.3 pulled into ATS core
  - Compatible with lua 5.1.x
  - Luajit has better performance
    - But a 2GB limit on memory used by lua VM

# Unit Test & Code Coverage

- Needs lunit & luacov
- Provide mock/stub framework

# Unit Test & Code Coverage

```
require "lunit"
require "luacov"

local mock = require ‘mock’
ts = require 'ts‘
ts.setup(mock)

local cors_lib = require ‘cors_lib'

module( “cors_testcase", lunit.testcase )

function test_failure()
  mock.client_request_header[‘Referer’] = ‘http://news.yahoo.com’
  cors_lib.execute()
  assert_equal(‘http://news.yahoo.com’,
           mock.client_response_header[‘Access-Control-Allow-Origin’],
           ‘Matched’)
end
```

# Memory Consumption

- 2GB limit due to luajit
- Check memory usage of your script
  - With the lua function - collectgarbage(“count”)
- If necessary, disable luajit
  - Build ATS with “--disable-luajit” option
  - Still need lua binary for plugin to work
  - Lua allows custom memory allocator when creating new VM to limit memory usage

# Performance

- Tested with 100K objects fetched 100 requests/s
- Latency of origin server is 500ms
- 3 Test cases
  - No plugin
  - header_rewrite plugin adding request/response headers
  - Lua script doing the same thing as above
- No effect on CPU / Memory utilization
- No effect on latency

“With Great Power Comes Great Responsbility”

# XSS

```
-- /test?<script>alert('XSS');</script>
function send_data()
  local nt = '<!DOCTYPE html><html lang="en-us"><head><title>Test</title>'..
             '</head><body>'..ts.ctx['args']..
             '</body></html>'

  local resp = 'HTTP/1.1 500 Internal Server Error\r\n' ..
    'Content-Type: text/html\r\n' ..
    'Content-Length: ' .. (string.len(nt)) .. '\r\n' ..
    'Last-Modified: '..os.date("%a, %d %b %Y %H:%M:%S GMT",os.time())..'\r\n'..
    'Connection: keep-alive\r\n' ..
    'Cache-Control: max-age=0, private\r\n\r\n' ..
    nt

  ts.say(resp)
end

function do_global_read_request()
  local args = ts.client_request.get_uri_args()
  ts.ctx[‘args’] = args
  ts.http.intercept(send_data)
end
```

# File System Attack

```
-- /test?test.lua
function do_global_read_request()
  local args = ts.client_request.get_uri_args()
  local f = io.open(args)
  local result = f:read(“*a”)
  f:close()

  ts.debug(result)
end
```

# Local File Inclusion

```
-- /test?test.lua
function do_global_read_request()
  local args = ts.client_request.get_uri_args()
  dofile(“/tmp/”..args)
end
```

# Code Injection

```
-- /test?os.clock()
function do_global_read_request()
  local args = ts.client_request.get_uri_args()
  loadstring(args)()
end
```

# OS Command Injection

```
-- /test?john
function do_global_read_request()
  local args = ts.client_request.get_uri_args()
  os.execute(“ls -l /home/”..args)
end
```

# Potential Security Concerns

| Name of the Attack | CWE (Common Weakness Enumeration) | Possible Way to Prevent |
|---|---|---|
| XSS | CWE-79 | Input Validation |
| SQL Injection | CWE-89 | Input Validation |
| File System Attack | N/A | Don’t pass user input to file system functions |
| File Inclusion | CWE-98 (for Lua as well) | Don’t pass user input to require() or dofile() |
| Code Injection | CWE-94 | Disable loadstring() |
| OS Command Injection | CWE-78 | Disable os.execute() & io.popen() |
| | | |

# Input/Output Validation

- Important to prevent many security problems
- Need to provide a library of functions

# Sandboxing

- Also important to disable some lua functions
- Sandboxing can be done in global or functional scope

```
function test()
  -- setting local environment for this function
  local env = {
    os = {clock = os.clock},
    string = {find = string.find}
  }
  setfenv (1, env)

  print(os.clock())
  -- cannot run this “os.execute(‘ls’)”

end
```

# What’s Next?

# Future

- Expose more ATS APIs as Lua functions
- Input Validation Library
- Open Source Mock/Stub framework

# TS-2281: Augment Config System to use LUA

## Pseudo Code

```
module(..., package.seeall)

function config()
  ats.config.proxy.config.http.server_ports("8888 8443:ipv4:ssl")
  ats.config.proxy.config.url_remap.pristine_host_hdr(0)
end

function config_ssl(add)
  add({ dest_ip = "*", ssl_cert_name = "bar.pem", ssl_key_name = "barKey.pem" })
end

function config_remap(r)
  r:definefilter("name", { src_ip = { "192.168.0.0-192.168.255.255",
                                      "10.0.0.0-10.255.255.255" }
                         , method = { "PURGE", "DELETE" }
                         , action = "allow" })
  r:activatefilter("name")
  r:map("http://localhost:8888/special/", "http://example.com/",
        { src_ip = "127.0.0.1" },
        { plugin = "foo.so", pparams = { "arg1", "arg2" } } )
  r:deactivatefilter("name")
end
```


Q & A

Thank You!

# Reference


1) ATS - https://docs.trafficserver.apache.org/en/latest/sdk/how-to-create-trafficserver-plugins.en.html
2) Lua - http://www.lua.org/
3) Computational Continuations - http://www.jquigley.com/files/talks/continuations.pdf
4) header_rewrite - https://docs.trafficserver.apache.org/en/latest/reference/plugins/header_rewrite.en.html
5) Varnish - https://www.varnish-cache.org/
6) VCL - https://www.varnish-cache.org/docs/4.0/users-guide/vcl.html
7) http://www.slideshare.net/bryan_call/choosing-a-proxy-server-apachecon-2014 (Slides 47)
8) http://www.bizety.com/2014/06/11/interview-founder-of-varnish-software/

# Reference


9) mod_lua - http://www.modlua.org/
10) ngx_lua - https://github.com/openresty/lua-nginx-module
11) Redis - http://redis.io/
12) Mysql-proxy - https://launchpad.net/mysql-proxy
13) Lua/World of Warcraft - http://www.wowwiki.com/Lua
14) ATS Lua Pugin Documentation - https://docs.trafficserver.apache.org/en/latest/reference/plugins/ts_lua.en.html
15) Luajit - http://luajit.org/luajit.html
16) Lunit - http://www.mroth.net/lunit/
17) Luacov - http://luacov.luaforge.net/
18) Custom memory allocator - http://stackoverflow.com/questions/9671793/limiting-a-lua-scripts-memory-usage
19) Lua Web Application Security Vulnerabilities - http://seclists.org/fulldisclosure/2014/May/128
20) TS-2281 - https://issues.apache.org/jira/browse/TS-2281