Lua’s coroutines: the secret sauce in Nmap’s Scripting Engine

Lua Workshop 2017

Patrick Donnelly
Software Engineer
2017 October 16th
Patrick Donnelly

- Started as a WoW Addon lurker on lua-l: batrick@batbytes.com
- Got excited finding a few bugs in Lua which led to...
- Outside of Lua: went to graduate school at University of Notre Dame for a PhD and now work on the Ceph file system at Red Hat.
What’s Nmap?

- Massively parallel network reconnaissance tool to find:
  - online hosts
  - open ports
  - OS running on hosts
  - network layout/security
  - everything there is to know about the server (NSE!)

```
$ nmap -p 80 --script 'safe and default' www.google.com
Starting Nmap 7.60 (https://nmap.org) at 2017-10-14 17:44 PDT
Nmap scan report for www.google.com (216.58.216.4)
Host is up (0.025s latency).
Other addresses for www.google.com (not scanned): 2607:f8b0:4007:804::2004
rDNS record for 216.58.216.4: lax02s21-in-f4.1e100.net

PORT     STATE SERVICE
80/tcp    open    http
| http-robots.txt: 214 disallowed entries (15 shown)
| /search /sdch /groups /index.html? /? /?hl=*&
|_/?hl=*&**&gws_rd=ssl /imgres /u/ /preferences /setprefs /default /m? /m/ /wml?
|_/http-title: Google
```

Nmap done: 1 IP address (1 host up) scanned in 1.33 seconds
Nmap Scripting Engine

- Parallel network script execution framework
- Scripts execute concurrently performing advanced and specific network reconnaissance against the host.
- NSE Includes hundreds of scripts and libraries
- Started as a GSOC project by Diman Todorov in 2008
Example Script

local http, stdnse = require "http", require "stdnse"

description, author = "get the web page title", "Patrick"

categories = {"default", "safe"}

function portrule(host, port)  return port.number == 80 end

function action(host, port)

  local resp = http.get( host, port, stdnse.get_script_args(SCRIPT_NAME..".url") or "/" )

  local title = string.match(resp.body, "<[Tt][Ii][Tt][Ll][Ee][^>]*>([^<]*)</[Tt][Ii][Tt][Ll][Ee]>")

  return title

end
$ nmap -p 80 --script $(pwd)/http-title.nse www.google.com

Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-08 21:18 PDT

[...]

Nmap scan report for www.google.com (216.58.217.196)

Host is up (0.022s latency).

[...]

PORT   STATE SERVICE

80/tcp open  http

|_http-title: Google

Nmap done: 1 IP address (1 host up) scanned in 0.46 seconds
Concurrent Script Execution

- Each script is instantiated in a coroutine which tests (rule function) the host and then gathers information (action)
- Script threads are run concurrently, network I/O causes scripts to yield

Nmap's NSOCK Asynchronous Networking Library

http-title

coroutine.resume

coroutine.yield

sock:read()
Challenge: Build an OS

But can it run emacs?

- NSE maintains tables of pending/waiting scripts.
- Also, has a generator which produces the next scripts to run (to limit the number of active scripts)
- Defines mechanisms for libraries to yield and restart scripts (used by nsock to resume a script once data is available)
- Most of it is in nse_main.lua (1500LOC)
- NSOCK limits script parallelism (limits # of active sockets / see also `--max-parallelism`)
Challenge: Mutual Exclusion

Mutual exclusion for coroutines? ... Why

- Scripts sometimes need to limit concurrency doing network operations
- First instance of the problem: whois script which gets WHOIS data from IANA servers. Doing WHOIS lookups for hundreds of target hosts results in getting banned. Who knew?

```
$ nmap -p 80 --script whois-domain.nse www.google.com

[...]

80/tcp open  http

Host script results:

| whois-domain: |
| Domain name record found at whois.verisign-grs.com |
| Domain Name: GOOGLE.COM
| Registry Domain ID: 2138514_DOMAIN_COM-VRSN
| Registrar WHOIS Server: whois.markmonitor.com
| Registrar URL: http://www.markmonitor.com
| Updated Date: 2011-07-20T16:55:31Z
| Creation Date: 1997-09-15T04:00:00Z

[...]
NSE Mutexes

mutex = stdnse.mutex(obj)
-- e.g. stdnse.mutex "my-script.x-critical-section"
mutex "lock"
mutex "done" -- yes, I feel bad about that argument name
mutex "trylock" -- joke, no one ever uses this
Where are Mutexes used?

- whois-*nse to serialize lookups
- http cached GETs, to avoid concurrent GETs of the same page
- Preventing concurrent SSL cert lookup + caching
- HTTP “slowloris” attack; only one attack at a time
Challenge: Multitasking Scripts

Oh no, we’re actually building a kernel...

- By default, a script is limited to doing one nsock operation at a time which prevents parallel network operations.
- Use case: http-spider library that does parallel GET requests against a target www server.
local httpspider = require "httpspider"
action = function(host, port)
    local maxpages = stdnse.get_script_args(SCRIPT_NAME .. " .maxpagecount") or 1
    local tries = stdnse.get_script_args(SCRIPT_NAME .. " .tries") or 5

    local dump = {}
    local crawler = httpspider.Crawler:new( host, port, nil, { scriptname = SCRIPT_NAME, maxpagecount = tonumber(maxpages) } )
    crawler:set_timeout(10000)
    -- launch the crawler
    while(true) do
        local start = stdnse.clock_ms()
        local status, r = crawler:crawl()
        if ( not(status) ) then break end
        local chrono = stdnse.clock_ms() - start
        dump[chrono] = tostring(r.url)
    end
    -- More processing...
end

Create the spider

Fetch a URI / crawl the website
### Script: http-chrono

- **PORT** | **STATE** | **SERVICE**
- 80/tcp | open | http

_Every request on the server is being monitored to track its performance._

- **http-chrono:** Request times for `/`; **avg:** 2.98ms; **min:** 2.63ms; **max:** 3.62ms

<table>
<thead>
<tr>
<th>PORT</th>
<th>STATE SERVICE</th>
<th>SCRIPT: http-chrono</th>
</tr>
</thead>
<tbody>
<tr>
<td>80/tcp</td>
<td>open</td>
<td>http</td>
</tr>
<tr>
<td></td>
<td></td>
<td>http-chrono:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>page</td>
</tr>
<tr>
<td></td>
<td></td>
<td>avg</td>
</tr>
<tr>
<td>/admin/</td>
<td></td>
<td>1.91ms</td>
</tr>
<tr>
<td>/manager/status</td>
<td></td>
<td>2.14ms</td>
</tr>
<tr>
<td>/manager/html</td>
<td></td>
<td>2.26ms</td>
</tr>
<tr>
<td>/examples/servlets/</td>
<td></td>
<td>2.43ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PORT</th>
<th>STATE SERVICE</th>
<th>SCRIPT: http-chrono</th>
</tr>
</thead>
<tbody>
<tr>
<td>80/tcp</td>
<td>open</td>
<td>http</td>
</tr>
<tr>
<td></td>
<td></td>
<td>http-chrono:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>page</td>
</tr>
<tr>
<td></td>
<td></td>
<td>avg</td>
</tr>
<tr>
<td>/admin/</td>
<td></td>
<td>1.91ms</td>
</tr>
<tr>
<td>/manager/status</td>
<td></td>
<td>2.14ms</td>
</tr>
<tr>
<td>/manager/html</td>
<td></td>
<td>2.26ms</td>
</tr>
<tr>
<td>/examples/servlets/</td>
<td></td>
<td>2.43ms</td>
</tr>
</tbody>
</table>

*Found these URIs for you using the spider library*
Example parallel HTTP GET function:

```lua
local http = require "http"
function pget(host, port, urls)
    local threads, responses = {}, {}
    local function do_get(i)
        responses[i] = http.get(host, port, urls[i])
    end
    for i = 1, #urls do
        threads[#threads+1] = stdnse.new_thread(do_get, i)
    end
    return responses
end
```

Launch concurrent thread

No memory sync needed

Does it work? NO
Challenge: Thread Synchronization

- How do we get scripts to coordinate with each other?
- Well, let’s borrow from another synchronization primitive... condition variables.

Is this presentation over?

Nmap’s NSOCK Asynchronous Networking Library
NSE Condition Variables

condvar = stdnse.condvar(obj)

-- e.g. local thread_pool = {}; stdnse.condvar(thread_pool)
condvar “wait” -- wait to be woken up
condvar “signal” -- wake up a sleeper
condvar “broadcast” -- wake up everyone
fixed parallel http get function:

local http = require "http"
function pget(host, port, urls)
    local threads, responses = {}, {}
    local condvar = stdnse.condvar(threads)

    local function do_get(i)
        responses[i] = http.get(host, port, urls[i])
        condvar "signal"
    end

    for i = 1, #urls do
        threads[#threads+1] = stdnse.new_thread(do_get, i)
    end

    repeat
        condvar "wait"
        for i, thread in ipairs(threads) do
            if coroutine.status(thread) == "dead" or responses[i] then
                threads[i] = nil
            end
        end
    until next(threads) == nil

    return responses
end

does it work? yes, but:
- should use worker thread pool
- needs error checking
Challenge: Coroutine Stacks (or “nested”)

- Yields across multiple threads requires changes to Lua’s coroutine library.

```lua
http.get(host, port, url)
for url in scrape(url) do
    http.get(host, port, url)
end
 coroutine.resume(script)
```

<table>
<thead>
<tr>
<th>NSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSock</td>
</tr>
<tr>
<td>scraper.lua</td>
</tr>
<tr>
<td>http-spider</td>
</tr>
</tbody>
</table>

int nse_yield (lua_State *L, lua_KContext ctx, lua_KFunction k)
{
    lua_getfield(L, LUA_REGISTRYINDEX, NSE_YIELD);
    lua_pushthread(L);
    lua_call(L, 1, 1); /* returns NSE_YIELD_VALUE */
    return lua_yieldk(L, 1, ctx, k); /* yield with NSE_YIELD_VALUE */
}

Local NSE_YIELD_VALUE = {}
local function handle (co, status, ...)
    if status and NSE_YIELD_VALUE == ... then -- NSE has yielded the thread
        return handle(co, resume(co, yield(NSE_YIELD_VALUE)));
    else
        return status, ...;
    end
end

function coroutine.resume (co, ...)
    return handle(co, resume(co, ...));
end

Called by nsock library

See also recent discussion on lua-l:

Defined by NSE. New coroutine.resume called by scripts/libraries
NSE Base Thread

**nse_main.cc:**

```c
void nse_base (lua_State *L) {
    lua_getfield(L, LUA_REGISTRYINDEX, NSE_BASE);
    lua_call(L, 0, 1); /* returns base thread */
}
```

**nse_main.lua:**

```lua
_R[BASE] = function ()
    return current and current.co;
end
```

**nse_nsock.cc:**

```c
static int socket_lock (lua_State *L, int idx) {
    unsigned p = o.max_parallelism == 0 ? MAX_PARALLELISM : o.max_parallelism;
    int top = lua_gettop(L);
    nse_base(L);
    lua_rawget(L, THREAD_SOCKETS);
}
```
But re-implementing in Lua code is so fun!
- Idea: we’d like to link to libssh2 to run scripts against ssh servers
- Problem: how do we get libssh2 to play nice with other scripts by using our networking framework?
- Naive solution/surrender: just accept ssh sessions block the process

#include <libssh2.h>

int libssh2_session_handshake(
    LIBSSH2_SESSION *session,
    libssh2_socket_t socket)
Solution: Give SSH a UNIX socket

- Each ssh session allocates a socketpair
- SSH gets one end of the socket to talk to non-blocking
- Benefit: we now get network I/O parallelism when using libssh2.

Cool! We’re calling methods on a socket userdata that may yield from C++! (lua_callk)
Conclusion: coroutines are awesome!

Patrick Donnelly
pdonnell@redhat.com
https://nmap.org/