lua Documentation

Release 0.0.1

Thomas Heloin

Jun 12, 2017

Contents:

You are here today to learn how you can use LUA in TShark to process large files. You will also learn how you can pre-process a log files to help you extract the right data

Capture and SSL key can be found here https://jumpshare.com/v/VMlpq669J7nxKQO7zhHw

CHAPTER 1

How to start LUA in TShark

This is your first lab to test Wireshark and Lua.

Lua script. Save it as hello.lua:

```
print("Hello World!")
```

Use an empty capture file, the -q is here to quiet TShark, so that it does not display the packets that it is processing.:

```
tshark -X lua_script:hello.lua -r empty.cap -q
```

CHAPTER 2

Capture splitter (new title)

Create a streams/ subfolder, the script does not create that folder for you and it will not work without it.

Tip: Use this command if you create too many files (Linux/Mac) in streams folder:

```
ls | xargs -n 100 rm
```

Tip: Raise your ulimit if your capture is really large:

```
ulimit -n 10000
```

Script:

```
-- Register the field we want Wireshark to tell us about, here we just want the TCP␣
↪stream ID
local tcp_stream = Field.new("tcp.stream")

-- Create a new "Listener" and the display filter "tcp", since we want to split tcp␣
↪streams out of the capture file.
local tap = Listener.new(nil,"tcp")

-- Create an array to store all of our file descriptors
local dumpers = {}

-- It might be prudent to raise your max file descriptor in an Unix system
-- ulimit -n 10000
-- https://superuser.com/questions/302754/increase-the-maximum-number-of-open-file-
↪descriptors-in-snow-leopard

-- Create a write packet function, which will take a stream id, and create a␣
↪corresponding file under a subfolder streams/
-- You will need to create that folder before you run the script
```

```
local function write_pkt(id)
        local file = dumpers[id]
        if not file then
                -- Dumper.new is a function of LUA for Wireshark, it will create a␣
↪capture file for us
                file = Dumper.new("streams/" .. id .. ".cap")

                -- There is a little complexity here, but essentially what we are␣
↪working around is a problem when you run out of file descriptors
                -- So we flush all open file descriptors, and start anew.
                if (file == nil) then
                        for file,dumper in pairs(dumpers) do
                                dumper:flush()
                                dumper:close()
                        end
                        dumpers = {}
                        file = assert(Dumper.new("streams/" .. id .. ".cap"))
                end
                dumpers[id] = file
        end

        -- Simply dump the current packet to our file
        file:dump_current()
end

-- tap.packet is a function called by Wireshark for every packet matching our listener
function tap.packet(pinfo,tvb,tapdata)
        write_pkt(tostring(tcp_stream()))
end

-- a listener tap's draw function is called every few seconds in the GUI
-- and at end of file (once) in TShark
function tap.draw()
        print("file processed, closing all dumpers")
        for file,dumper in pairs(dumpers) do
                dumper:flush()
                dumper:close()
        end
        dumpers = {}
end

-- a listener tap's reset function is called at the end of a live capture run,
-- when a file is opened, or closed.  TShark never appears to call it.
function tap.reset()
end
```

Exercise 1

Create your own capture file and extract all traffic that is not TCP.

Exercise 2

Only split traffic where the destination is TCP port 80

CHAPTER 3

Extracting some HTTP

The following LUA script is a little more complex, we will now extract HTTP request and response from our capture file and split them per TCP streams.

Note: You can use the 0.cap previously created as source to try out your code.:

```
tshark -X lua_script:http.lua -r streams/0.cap -q
```

Using other streams is unlikely to work, try to workout why.

After execution, you should have a whole bunch of S-<id>.txt in your streams folder

Script:

```
-- Let's register all the fields we want TShark to extract
-- Remember that we are creating functions retrieving the field values.

local tcp_stream = Field.new("tcp.stream")
local http_method = Field.new("http.request.method")
local http_uri = Field.new("http.request.uri")
local http_version = Field.new("http.request.version")
local http_code = Field.new("http.response.code")
local http_phrase = Field.new("http.response.phrase")
local http_location = Field.new("http.location")
local http_request = Field.new("http.request")
local http_response = Field.new("http.response")
local http_cookie = Field.new("http.cookie_pair")
local http_setcookie = Field.new("http.set_cookie")
local http_request_header = Field.new("http.request.line")
local http_response_header = Field.new("http.response.line")
local http_response_data = Field.new("http.file_data")

-- Creating the listener, to avoid http data appearing twice due to retransmit, let's␣
↪suppress them
local tap = Listener.new(nil,"http && !tcp.analysis.retransmission && !tcp.analysis.
↪lost_segment")
```

```
local dumpers = {}

-- To avoid runtime error due to nil variable, this function will simply display␣
↪nothing if TShark has not found the field we are looking for.
local function to_string(string)
        if (string == nil) then
                return ""
        else
                return tostring(string)
        end
end

-- Same as the previous example, but this time we are writing text with normal LUA I/
↪O functions.
local function write_msg(id,msg)
        local file = dumpers[id]
        if not file then
                file = io.open("streams/" .. id .. ".txt", "a")
                if (file == nil) then
                        for file,dumper in pairs(dumpers) do
                                dumper:flush()
                                dumper:close()
                        end
                        dumpers = {}
                        file = assert(io.open("streams/" .. id .. ".txt", "a"))
                end
                dumpers[id] = file
        end
        file:write(msg)
end

function tap.packet(pinfo,tvb,tapdata)

        -- The only reason I am checking if this is a http request is due to the␣
↪cookies
        if ( http_request() ) then
                if( http_method() == nil ) then return end

                local request_mrhsession = ""
                local cookie = {http_cookie()}
                for i in pairs(cookie) do
                        -- One of the powerful function of LUA is its text matching
                        request_mrhsession = tostring(cookie[i]):match '.*LastMRH_
↪Session=(%w+).*'
                        if (request_mrhsession ~= nil ) then break end
                end

                -- pinfo is from our tap, and contains packet information created by␣
↪Wireshark, like timestamps.
                local msg =
                        to_string("** " .. string.format("%8d", pinfo.number)) .. "␣
↪** " ..
                        to_string(format_date(pinfo.abs_ts)) .. " " ..
                        to_string(http_method()) .. " " ..
                        to_string(http_uri()) .. " " ..
                        to_string(http_version()) .. " " ..
                        to_string(request_mrhsession) .. " " ..
```

```
                        to_string(tcp_stream())
                        .. "\n"

                -- Writing to file all of our HTTP request headers, we are adding a
↪"S-" prefix to the filename
                write_msg("S-" .. tostring(tcp_stream()),msg)
                local header = {http_request_header()}
                for i in pairs(header) do
                        write_msg("S-" .. tostring(tcp_stream())," " ..␣
↪tostring(header[i]))
                end

        else

                -- If that is not a request, then it is a response :)
                local response_mrhsession = ""
                local cookie = {http_setcookie()}
                for i in pairs(cookie) do
                        response_mrhsession = tostring(cookie[i]):match '.*LastMRH_
↪Session=(%w+).*'
                        if (response_mrhsession ~= nil ) then break end
                end

                local msg =
                        to_string("** " .. string.format("%8d", pinfo.number)) .. "␣
↪** " ..
                        to_string(format_date(pinfo.abs_ts)) .. " " ..
                        to_string(http_code()) .. " " ..
                        to_string(http_phrase()) .. " " ..
                        to_string(http_version()) .. " " ..
                        to_string(response_mrhsession) .. " " ..
                        to_string(http_location()) .. " " ..
                        to_string(tcp_stream())
                        .. "\n"

                write_msg("S-" .. tostring(tcp_stream()),msg)
                local header = {http_response_header()}
                for i in pairs(header) do
                        write_msg("S-" .. tostring(tcp_stream())," " ..␣
↪tostring(header[i]))
                end
        end
end

-- a listener tap's draw function is called every few seconds in the GUI
-- and at end of file (once) in tshark
function tap.draw()
        print("file processed, closing all dumpers")
        for file,dumper in pairs(dumpers) do
                dumper:flush()
                dumper:close()
        end
        dumpers = {}
end

-- a listener tap's reset function is called at the end of a live capture run,
-- when a file is opened, or closed.  Tshark never appears to call it.
function tap.reset()
```

```
end
```

Exercise 1

Merge the previous script with this one. Nothing stops you from doing 2 things at the same time!

Exercise 2

What can you do in Wireshark to help you decrypt other streams but 0?

Exercise 3

Can you add the whole http response to your files?

CHAPTER 4

Reading a log file to add information to your streams

We are now going a open an APM log file, extract session ID value and usernames.

Script:

```
-- Let's register all the fields we want TShark to extract
-- Remember that we are creating functions retrieving the field values.

local tcp_stream = Field.new("tcp.stream")
local http_method = Field.new("http.request.method")
local http_uri = Field.new("http.request.uri")
local http_version = Field.new("http.request.version")
local http_code = Field.new("http.response.code")
local http_phrase = Field.new("http.response.phrase")
local http_location = Field.new("http.location")
local http_request = Field.new("http.request")
local http_response = Field.new("http.response")
local http_cookie = Field.new("http.cookie_pair")
local http_setcookie = Field.new("http.set_cookie")
local http_request_header = Field.new("http.request.line")
local http_response_header = Field.new("http.response.line")
local http_response_data = Field.new("http.file_data")

-- Creating the listener, to avoid http data appearing twice due to retransmit, let's␣
↪suppress them
local tap = Listener.new(nil,"http && !tcp.analysis.retransmission && !tcp.analysis.
↪lost_segment")

local dumpers = {}

-- To avoid runtime error due to nil variable, this function will simply display␣
↪nothing if TShark has not found the field we are looking for.
local function to_string(string)
        if (string == nil) then
                return ""
        else
```

```
                return tostring(string)
        end
end

-- Same as the previous example, but this time we are writing text with normal LUA I/
↪O functions.
local function write_msg(id,msg)
        local file = dumpers[id]
        if not file then
                file = io.open("streams/" .. id .. ".txt", "a")
                if (file == nil) then
                        for file,dumper in pairs(dumpers) do
                                dumper:flush()
                                dumper:close()
                        end
                        dumpers = {}
                        file = assert(io.open("streams/" .. id .. ".txt", "a"))
                end
                dumpers[id] = file
        end
        file:write(msg)
end

-- Let's create some arrays that will be useful when reading our APM log files.
local timeouts = {}
local months = {}
local maps = {}

-- Need to convert months as found in the log file to something LUA will understand
months["Jan"] = 1
months["Feb"] = 2
months["Mar"] = 3
months["Apr"] = 4
months["May"] = 5
months["Jun"] = 6
months["Jul"] = 7
months["Aug"] = 8
months["Sep"] = 9
months["Oct"] = 10
months["Nov"] = 11
months["Dec"] = 12

local apm = assert(io.open("apm", "r"))
for l in apm:lines() do
        -- Jun  3 19:31:15 pulsar notice tmm1[10766]: 01490520:5: /Common/
↪test:Common:216342fe: Session deleted due to admin initiated termination.
        local month, day, h, m, s, session, reason = l:match '(%w+) (%d+) (%d+):(
↪%d+):(%d+).*Common:(%w+): Session deleted due to (%w+) .*'
        if (session ~= nil) then
                -- There is no year to extract from the log file
                local convertTime = os.time({year = "2017", month = months[month],␣
↪day = day, hour = h, min = m, sec = s})
                -- You may need to offset the timestamps
                -- convertTime = convertTime - 3600

                -- Adding the expiry timestamp (and reason) for a given session to␣
↪our table
                timeouts[session] = {convertTime, reason}
```

```
        end
        -- Jun  3 19:30:13 pulsar notice apmd[6566]: 01490010:5: /Common/
↪test:Common:ab9bee05: Username 'u-0088979'
        local session, username = l:match '.*Common:(%w+): Username \'(.*)\''
        if (username ~= nil) then
                -- Adding the our username for a given session to our table,␣
↪sometimes sessions are created with no Username, so lets ignore those.
                maps[session] = username
        end
end
print("Done processing APM log file")

function tap.packet(pinfo,tvb,tapdata)

        -- The only reason I am checking if this is a http request is due to the␣
↪cookies
        if ( http_request() ) then
                if( http_method() == nil ) then return end

                local request_mrhsession = ""
                local cookie = {http_cookie()}
                for i in pairs(cookie) do
                        -- One of the powerful function of LUA is its text matching
                        request_mrhsession = tostring(cookie[i]):match '.*LastMRH_
↪Session=(%w+).*'
                        if (request_mrhsession ~= nil ) then break end
                end

                -- What we are doing here is to compare the current timestamp of our␣
↪packet with an APM session
                local username = ""
                if ( request_mrhsession ~= nil and maps[request_mrhsession] ~= nil )␣
↪then

                        -- retrieve the username from the LastMRH_Session cookie
                        username = maps[request_mrhsession]

                        -- do we know when this session was expired
                        if(timeouts[request_mrhsession] ~= nil) then

                                -- If the packet timestamps is superior, then the␣
↪session has been previously deleted, and that user created a new session
                                if(tonumber(pinfo.abs_ts) > tonumber(timeouts[request_
↪mrhsession][1])) then
                                        local msg = "** " .. string.format("%8d",␣
↪pinfo.number) .. " ** " .. os.date("%b %d, %Y %X",timeouts[request_mrhsession][1]) .
↪. ".000000000 PDT +++++++ THIS USER HAS HAD HIS SESSION TERMINATED BY " ..␣
↪timeouts[request_mrhsession][2] .. "\n"
                                        write_msg(username,msg)
                                        -- Lets remove that key from our map table,␣
↪otherwise we will keep writing the message above all the time
                                        timeouts[request_mrhsession] = nil
                                end
                        end
                end

                local msg =
                        to_string("** " .. string.format("%8d", pinfo.number)) .. "␣
↪** " ..
```

```
                        to_string(format_date(pinfo.abs_ts)) .. " " ..
                        to_string(http_method()) .. " " ..
                        to_string(http_uri()) .. " " ..
                        to_string(http_version()) .. " " ..
                        to_string(request_mrhsession) .. " " ..
                        to_string(username) .. " " ..
                        to_string(tcp_stream())
                        .. "\n"

                -- Writing to file the request for a given username, since we know it.
↪ That been said, there is a limit here an initial request with no MRH will not␣
↪match.
                -- We would need to wait for the response and then retrieve the␣
↪corresponding request to finally write it to file.
                -- RFE: Need to store the request in memory with the stream ID, and␣
↪probably a request number. Wait for the matching response and commit to file.
                if ( username ~= "" ) then
                        write_msg(username,msg)
                else
                        write_msg("u-UNKNOWN",msg)
                end

                -- Writing to file all of our HTTP request headers, we are adding a
↪"S-" prefix to the filename
                write_msg("S-" .. tostring(tcp_stream()),msg)
                local header = {http_request_header()}
                for i in pairs(header) do
                        write_msg("S-" .. tostring(tcp_stream())," " ..␣
↪tostring(header[i]))
                end

        else

                -- If that is not a request, then it is a response :)
                local response_mrhsession = ""
                local cookie = {http_setcookie()}
                for i in pairs(cookie) do
                        response_mrhsession = tostring(cookie[i]):match '.*LastMRH_
↪Session=(%w+).*'
                        if (response_mrhsession ~= nil ) then break end
                end

                local username = ""
                if ( response_mrhsession ~= nil and maps[response_mrhsession] ~= nil␣
↪) then
                        username = maps[response_mrhsession]
                end

                local msg =
                        to_string("** " .. string.format("%8d", pinfo.number)) .. "␣
↪** " ..
                        to_string(format_date(pinfo.abs_ts)) .. " " ..
                        to_string(http_code()) .. " " ..
                        to_string(http_phrase()) .. " " ..
                        to_string(http_version()) .. " " ..
                        to_string(response_mrhsession) .. " " ..
                        to_string(username) .. " " ..
                        to_string(http_location()) .. " " ..
```

```
                        to_string(tcp_stream())
                        .. "\n"

                if ( username ~= "" ) then
                        write_msg(username,msg)
                else
                        write_msg("u-UNKNOWN",msg)
                end

                write_msg("S-" .. tostring(tcp_stream()),msg)
                local header = {http_response_header()}
                for i in pairs(header) do
                        write_msg("S-" .. tostring(tcp_stream())," " ..␣
↪tostring(header[i]))
                end
                -- write_msg("S-" .. tostring(tcp_stream()),to_string(http_response_
↪data()))
        end
end

-- a listener tap's draw function is called every few seconds in the GUI
-- and at end of file (once) in tshark
function tap.draw()
        print("file processed, closing all dumpers")
        for file,dumper in pairs(dumpers) do
                dumper:flush()
                dumper:close()
        end
        dumpers = {}

        print("adding last user timeouts to log files")
        for key,value in pairs(timeouts) do
                local username = maps[key]
                if (username ~= nil) then
                        local msg = "**        - ** " .. os.date("%b %d, %Y %X",
↪value[1]) .. ".000000000 PDT ####### THIS USER HAS HAD HIS SESSION TERMINATED BY " .
↪. value[2] .. "\n"
                        local file = io.open("streams/" .. username .. ".txt", "r")
                        if (file ~= nil) then
                                file:close()
                                local file = io.open("streams/" .. username .. ".txt",
↪ "a")
                                file:write(msg)
                                file:close()
                        end
                end
        end
end

-- a listener tap's reset function is called at the end of a live capture run,
-- when a file is opened, or closed.  Tshark never appears to call it.
function tap.reset()
end
```

Exercise 1

We are not adding the timeouts to our files, using your S-0.cap file try to workout why.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search