LAMPIRAN

Lampiran kode *scripting* Lua dari Wireshark untuk protokol menggunakan HTTP Response dan TCP Sequence Analyzer

```
-- Intro Information
local my_info = {
   name = "HTTP Response and Shortcut",
   version = "0.1 (Alpha)",
}
-- description: Scripting HTTP Response script lua plugin untuk Wireshark
-- Imput scripting
local http_response_proto = Proto("http_resp", "Upstream response HTTP request")
local http_shortcut_detail = Proto("http_shortcut", "Shortcut HTTP data")

local stream_map = {}  -- table per stream : { req_cnt, resp_cnt, reqs }
local resp_map = {}    -- pemetaan table dari respons (response) ke permintaan (request)

set_plugin_info(my_info)

-- Tcp Sequence Numbers Analyzer
-- Petunjuk: buka melalui terminal "wireshark -X lua_script:response.lua"

-- menu fungsi

function tsa_menu_func()
   -- tsa analyze function
   function tsa_do(ip1, ip2)
      local results = {
         -- Total
         ["counter"] = 0,
         ["dupack"] = 0,
         ["retrans"] = 0,
         ["fastretrans"] = 0,
         ["zerowindow"] = 0,
         ["windowfull"] = 0,

         -- dari ip1 ke ip2
         ["counter1to2"] = 0,
         ["dupack1to2"] = 0,
         ["retrans1to2"] = 0,
         ["fastretrans1to2"] = 0,
         ["zerowindow1to2"] = 0,
         ["windowfull1to2"] = 0,

         -- dari ip2 ke ip1
         ["counter2to1"] = 0,
```

```
        ["dupack2to1"] = 0,
        ["retrans2to1"] = 0,
        ["fastretrans2to1"] = 0,
        ["zerowindow2to1"] = 0,
        ["windowfull2to1"] = 0,
    }
    local result_win = TextWindow.new("Tcp Sequence Numbers Analyze")
    local http_port = 80

    -- pemanggilan tap.draw
    function refresh_result()
        result_win:clear()
        result_win:set("Total\n")
        result_win:append("\tPackets counter: " .. results["counter"] .. "\n")
        result_win:append("\tDuplicate ACK: " .. results["dupack"] .. "\n")
        result_win:append("\tRetransmission: " .. results["retrans"] .. "\n")
        result_win:append("\tFast Retransmission: " .. results["fastretrans"] .. "\n")
        result_win:append("\tZero Window: " .. results["zerowindow"] .. "\n")
        result_win:append("\tWindow Full: " .. results["windowfull"] .. "\n")
        result_win:append("\n")
        result_win:append(ip1 .. " -> " .. ip2 .. "\n")
        result_win:append("\tPackets counter: " .. results["counter1to2"] .. "\n")
        result_win:append("\tDuplicate ACK: " .. results["dupack1to2"] .. "\n")
        result_win:append("\tRetransmission: " .. results["retrans1to2"] .. "\n")
        result_win:append("\tFast Retransmission: " .. results["fastretrans1to2"] .. "\n")
        result_win:append("\tZero Window: " .. results["zerowindow1to2"] .. "\n")
        result_win:append("\tWindow Full: " .. results["windowfull1to2"] .. "\n")
        result_win:append("\n")
        result_win:append(ip2 .. " -> " .. ip1 .. "\n")
        result_win:append("\tPackets counter: " .. results["counter2to1"] .. "\n")
        result_win:append("\tDuplicate ACK: " .. results["dupack2to1"] .. "\n")
        result_win:append("\tRetransmission: " .. results["retrans2to1"] .. "\n")
        result_win:append("\tFast Retransmission: " .. results["fastretrans2to1"] .. "\n")
        result_win:append("\tZero Window: " .. results["zerowindow2to1"] .. "\n")
        result_win:append("\tWindow Full: " .. results["windowfull2to1"] .. "\n")
    end

    -- packets counter, total
    local counter_tap = Listener.new("frame", "ip.addr == " .. ip1 .. " && ip.addr == " .. ip2 .. "
&& tcp.port == " .. http_port)
    function counter_tap.reset()
        results["counter"] = 0
    end
    function counter_tap.packet(pinfo, tvb, ip)
        results["counter"] = results["counter"] + 1
    end
    function counter_tap.draw()
        refresh_result()
```

```
    end

    -- dupack, total
    local dupack_tap = Listener.new("frame", "ip.addr == " .. ip1 .. " && ip.addr == " .. ip2 .. " &
& tcp.analysis.duplicate_ack" .. " && tcp.port == " .. http_port)
    function dupack_tap.reset()
       results["dupack"] = 0
    end
    function dupack_tap.packet(pinfo, tvb, ip)
       results["dupack"] = results["dupack"] + 1
    end
    function dupack_tap.draw()
       refresh_result()
    end

    -- retransmission, total
    local retrans_tap = Listener.new("frame", "ip.addr == " .. ip1 .. " && ip.addr == " .. ip2 .. "
&& tcp.analysis.retransmission" .. " && tcp.port == " .. http_port)
    function retrans_tap.reset()
       results["retrans"] = 0
    end
    function retrans_tap.packet(pinfo, tvb, ip)
       results["retrans"] = results["retrans"] + 1
    end
    function retrans_tap.draw()
       refresh_result()
    end

    -- fast retransmission, total
    local fastretrans_tap = Listener.new("frame", "ip.addr == " .. ip1 .. " && ip.addr == " .. ip2 .
. " && tcp.analysis.fast_retransmission" .. " && tcp.port == " .. http_port)
    function fastretrans_tap.reset()
       results["fastretrans"] = 0
    end
    function fastretrans_tap.packet(pinfo, tvb, ip)
       results["fastretrans"] = results["fastretrans"] + 1
    end
    function fastretrans_tap.draw()
       refresh_result()
    end

    -- zero window, total
    local zerowindow_tap = Listener.new("frame", "ip.addr == " .. ip1 .. " && ip.addr == " .. ip2 ..
 " && tcp.analysis.zero_window" .. " && tcp.port == " .. http_port)
    function zerowindow_tap.reset()
       results["zerowindow"] = 0
    end
    function zerowindow_tap.packet(pinfo, tvb, ip)
```

```
        results["zerowindow"] = results["zerowindow"] + 1
    end
    function zerowindow_tap.draw()
        refresh_result()
    end

    -- window full, total
    local windowfull_tap = Listener.new("frame", "ip.addr == " .. ip1 .. " && ip.addr == " .. ip2 ..
 " && tcp.analysis.window_full" .. " && tcp.port == " .. http_port)
    function windowfull_tap.reset()
        results["windowfull"] = 0
    end
    function windowfull_tap.packet(pinfo, tvb, ip)
        results["windowfull"] = results["windowfull"] + 1
    end
    function windowfull_tap.draw()
        refresh_result()
    end

    -- packets counter, ip1 -> ip2
    local counter1to2_tap = Listener.new("frame", "ip.src == " .. ip1 .. " && ip.dst == " .. ip2 ..
" && tcp.port == " .. http_port)
    function counter1to2_tap.reset()
        results["counter1to2"] = 0
    end
    function counter1to2_tap.packet(pinfo, tvb, ip)
        results["counter1to2"] = results["counter1to2"] + 1
    end
    function counter1to2_tap.draw()
        refresh_result()
    end

    -- dupack, ip1 -> ip2
    local dupack1to2_tap = Listener.new("frame", "ip.src == " .. ip1 .. " && ip.dst == " .. ip2 .. "
 && tcp.analysis.duplicate_ack" .. " && tcp.port == " .. http_port)
    function dupack1to2_tap.reset()
        results["dupack1to2"] = 0
    end
    function dupack1to2_tap.packet(pinfo, tvb, ip)
        results["dupack1to2"] = results["dupack1to2"] + 1
    end
    function dupack1to2_tap.draw()
        refresh_result()
    end

    -- retransmission, ip1 -> ip2
    local retrans1to2_tap = Listener.new("frame", "ip.src == " .. ip1 .. " && ip.dst == " .. ip2 ..
" && tcp.analysis.retransmission" .. " && tcp.port == " .. http_port)
```

```
    function retrans1to2_tap.reset()
       results["retrans1to2"] = 0
    end
    function retrans1to2_tap.packet(pinfo, tvb, ip)
       results["retrans1to2"] = results["retrans1to2"] + 1
    end
    function retrans1to2_tap.draw()
       refresh_result()
    end

    -- fast retransmission, ip1 -> ip2
    local fastretrans1to2_tap = Listener.new("frame", "ip.src == " .. ip1 .. " && ip.dst == " .. ip2
 .. " && tcp.analysis.fast_retransmission" .. " && tcp.port == " .. http_port)
    function fastretrans1to2_tap.reset()
       results["fastretrans1to2"] = 0
    end
    function fastretrans1to2_tap.packet(pinfo, tvb, ip)
       results["fastretrans1to2"] = results["fastretrans1to2"] + 1
    end
    function fastretrans1to2_tap.draw()
       refresh_result()
    end

    -- zero window, ip1 -> ip2
    local zerowindow1to2_tap = Listener.new("frame", "ip.src == " .. ip1 .. " && ip.dst == " .. ip2
.. " && tcp.analysis.zero_window" .. " && tcp.port == " .. http_port)
    function zerowindow1to2_tap.reset()
       results["zerowindow1to2"] = 0
    end
    function zerowindow1to2_tap.packet(pinfo, tvb, ip)
       results["zerowindow1to2"] = results["zerowindow1to2"] + 1
    end
    function zerowindow1to2_tap.draw()
       refresh_result()
    end

    -- window full, ip1 -> ip2
    local windowfull1to2_tap = Listener.new("frame", "ip.src == " .. ip1 .. " && ip.dst == " .. ip2
.. " && tcp.analysis.window_full" .. " && tcp.port == " .. http_port)
    function windowfull1to2_tap.reset()
       results["windowfull1to2"] = 0
    end
    function windowfull1to2_tap.packet(pinfo, tvb, ip)
       results["windowfull1to2"] = results["windowfull1to2"] + 1
    end
    function windowfull1to2_tap.draw()
       refresh_result()
    end
```

```
    -- packets counter, ip2 -> ip1
    local counter2to1_tap = Listener.new("frame", "ip.src == " .. ip2 .. " && ip.dst == " .. ip1 ..
" && tcp.port == " .. http_port)
    function counter2to1_tap.reset()
       results["counter2to1"] = 0
    end
    function counter2to1_tap.packet(pinfo, tvb, ip)
       results["counter2to1"] = results["counter2to1"] + 1
    end
    function counter2to1_tap.draw()
       refresh_result()
    end

    -- dupack, ip2 -> ip1
    local dupack2to1_tap = Listener.new("frame", "ip.src == " .. ip2 .. " && ip.dst == " .. ip1 .. "
 && tcp.analysis.duplicate_ack" .. " && tcp.port == " .. http_port)
    function dupack2to1_tap.reset()
       results["dupack2to1"] = 0
    end
    function dupack2to1_tap.packet(pinfo, tvb, ip)
       results["dupack2to1"] = results["dupack2to1"] + 1
    end
    function dupack2to1_tap.draw()
       refresh_result()
    end

    -- retransmission, ip2 -> ip1
    local retrans2to1_tap = Listener.new("frame", "ip.src == " .. ip2 .. " && ip.dst == " .. ip1 ..
" && tcp.analysis.retransmission" .. " && tcp.port == " .. http_port)
    function retrans2to1_tap.reset()
       results["retrans2to1"] = 0
    end
    function retrans2to1_tap.packet(pinfo, tvb, ip)
       results["retrans2to1"] = results["retrans2to1"] + 1
    end
    function retrans2to1_tap.draw()
       refresh_result()
    end

    -- fast retransmission, ip2 -> ip1
    local fastretrans2to1_tap = Listener.new("frame", "ip.src == " .. ip2 .. " && ip.dst == " .. ip1
 .. " && tcp.analysis.fast_retransmission" .. " && tcp.port == " .. http_port)
    function fastretrans2to1_tap.reset()
       results["fastretrans2to1"] = 0
    end
    function fastretrans2to1_tap.packet(pinfo, tvb, ip)
       results["fastretrans2to1"] = results["fastretrans2to1"] + 1
```

```
    end
    function fastretrans2to1_tap.draw()
       refresh_result()
    end

    -- zero window, ip2 -> ip1
    local zerowindow2to1_tap = Listener.new("frame", "ip.src == " .. ip2 .. " && ip.dst == " .. ip1
.. " && tcp.analysis.zero_window" .. " && tcp.port == " .. http_port)
    function zerowindow2to1_tap.reset()
       results["zerowindow2to1"] = 0
    end
    function zerowindow2to1_tap.packet(pinfo, tvb, ip)
       results["zerowindow2to1"] = results["zerowindow2to1"] + 1
    end
    function zerowindow2to1_tap.draw()
       refresh_result()
    end

    -- window full, ip2 -> ip1
    local windowfull2to1_tap = Listener.new("frame", "ip.src == " .. ip2 .. " && ip.dst == " .. ip1
.. " && tcp.analysis.window_full" .. " && tcp.port == " .. http_port)
    function windowfull2to1_tap.reset()
       results["windowfull2to1"] = 0
    end
    function windowfull2to1_tap.packet(pinfo, tvb, ip)
       results["windowfull2to1"] = results["windowfull2to1"] + 1
    end
    function windowfull2to1_tap.draw()
       refresh_result()
    end

    function remove_alltap()
       counter_tap:remove()
       dupack_tap:remove()
       retrans_tap:remove()
       fastretrans_tap:remove()
       zerowindow_tap:remove()
       windowfull_tap:remove()

       counter1to2_tap:remove()
       dupack1to2_tap:remove()
       retrans1to2_tap:remove()
       fastretrans1to2_tap:remove()
       zerowindow1to2_tap:remove()
       windowfull1to2_tap:remove()

       counter2to1_tap:remove()
       dupack2to1_tap:remove()
```

```
        retrans2to1_tap:remove()
        fastretrans2to1_tap:remove()
        zerowindow2to1_tap:remove()
        windowfull2to1_tap:remove()
      end

      result_win:set_atclose(remove_alltap)

      -- tap ulang  semua paket, kemudian semuanya mulai bekerja
      retap_packets()

   end

   -- Prompt untuk ip address
   new_dialog("Masukkan alamat IP yang terhubung", tsa_do, "ip address 1:", "ip address 2:")
end

-- Tools untuk respons di tampilan tree
-- memasukkan untuk respons pada proto fields
http_response_proto.fields.re_req_method = ProtoField.string("http_resp.request.method", "Request Meth
od")
http_response_proto.fields.re_req_uri    = ProtoField.string("http_resp.request.uri", "Request URI")
http_response_proto.fields.re_req_ver    = ProtoField.string("http_resp.request.version", "Request Ver
sion")
http_response_proto.fields.re_host       = ProtoField.string("http_resp.host", "Host")

-- memasukkan untuk beberapa permintaan dan respons
http_shortcut_detail.fields.URL          = ProtoField.string("http.request.URL", "Request URL")

local f_req_meth    = Field.new("http.request.method")
local f_req_uri     = Field.new("http.request.uri")
local f_req_ver     = Field.new("http.request.version")
local f_req_host    = Field.new("http.host")
local f_resp_code   = Field.new("http.response.code")
local f_tcp_stream  = Field.new("tcp.stream")
local f_tcp_dstport = Field.new("tcp.dstport")
local f_tcp_srcport = Field.new("tcp.srcport")
local f_ip_dsthost  = Field.new("ip.dst_host")
local f_ip_srchost  = Field.new("ip.src_host")

 function http_response_proto.init()
     stream_map = {}
     resp_map = {}
 end

 local function optional_port(tcp_port)
     if tcp_port ~= 443 and tcp_port ~= 80 then
         return ":" .. tcp_port
```

```
    end
    return ""
end

local function scheme_by_port(tcp_port)
    if tcp_port == 443 then
        return "https://"
    else
        return "http://"
    end
end

function http_response_proto.dissector(tvbuffer, pinfo, treeitem)
      if not f_tcp_stream() then return end
      local stream_n
      local URL
      stream_n = f_tcp_stream().value
      URL = nil
      -- ketika mendapat permintaan (request) pertama kali, masukkan ke stream_map
      if f_req_meth() then
          tcp_port = f_tcp_dstport().value
          local host
          if f_req_host() then
              host = f_req_host().value
          else
              host = f_ip_dsthost().value
          end
          URL = scheme_by_port (tcp_port) .. host .. optional_port(tcp_port) .. f_req_uri().value
          if not pinfo.visited then
              if stream_map[stream_n] == nil then
                  stream_map[stream_n] = {0,0,{}}
              end
              -- perhitungan kedatangan permintaan
              request_n = stream_map[stream_n][1] + 1
              stream_map[stream_n][1] = request_n
              stream_map[stream_n][3][ request_n ] = {f_req_meth().value, f_req_uri().value, f_req_ve
r().value, f_req_host().value}
          end
      end

      -
- jika mendapat respons, arahkan respons stream dengan cara arus balik dan arahkan ke permintaan yang
sesuai
      if f_resp_code() then
          local response_n
          if not pinfo.visited then
              if not stream_map[stream_n] then
                  warn("HTTP response not in stream (" .. pinfo.number .. ")")
```

```
                response_n = 0
            else
                response_n = stream_map[stream_n][2] + 1
                stream_map[stream_n][2] = response_n
                resp_map[pinfo.number] = response_n
            end
        else
            response_n = resp_map[pinfo.number]
        end

        tcp_port = f_tcp_srcport().value
        if response_n > 0 then
            local subtree = treeitem:add(http_response_proto, nil)
            local data = stream_map[stream_n][3][response_n]
            if data then
                subtree:add(http_response_proto.fields.re_req_method, data[1]):set_generated()
                subtree:add(http_response_proto.fields.re_req_uri, data[2]):set_generated()
                subtree:add(http_response_proto.fields.re_req_ver, data[3]):set_generated()
                if (data[4]) then
                    subtree:add(http_response_proto.fields.re_host, data[4]):set_generated()
                end

                host = data[4]
                if not host then
                    host = f_ip_srchost().value
                end

                URL = scheme_by_port(tcp_port) .. host .. optional_port(tcp_port) .. data[2]
            else
                warn("HTTP request data lost (" .. stream_n .. "," .. response_n .. ")")
            end
        end
    end

  if URL then
      local extra_tree = treeitem:add(http_shortcut_detail, nil)
      extra_tree:add(http_shortcut_detail.fields.URL, URL):set_generated()
  end
  end

-- Register post-dissector
register_postdissector(http_response_proto)
register_menu("Tcp Sequence Numbers Analyze", tsa_menu_func, MENU_TOOLS_UNSORTED)
```

Lampiran screenshot Perbandingan *scripting* Lua kasus 1 dan kasus 2 dengan Statistic Endpoint bedasarkan hitungan paket