BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Pada tahap implementasi ini, penulis akan membahas tahapan-tahapan yang dilakukan dalam mengaktifkan inisialisasi Lua pada file. Setelah melakukan rancangan, selanjutnya membuat *scripting* lua, menjelaskan cara kerja fungsi dari Lua *dissector*, kemudian dievaluasi dengan kasus uji dalam bentuk kesimpulan.

Kasus uji menunjukkan proses kumpulan *dissector* berupa jumlah angka yang akan dieksekusi dari *IP* sumber (source) ke *IP* destinasi, kemudian ditampilkan ke dalam jendela TCP Sequence Number Analyzer. Selain itu, HTTP Response dan Request ditampilkan ke panel detail paket agar lebih ringkas dan mudah dibaca. Kasus uji akan diperkenalkan dengan akurasi sedemikian rupa sehingga hasilnya dapat diproses.

Sebelum menjalankan *scripting* lua, di Gambar 4.1 menjelaskan konfigurasi pertama berisi *scripting* untuk menjalankan perintah mengaktifkan dukungan lua, dan perintah untuk akses *super user*.

```
15. disable_lua = false
16.
17. if not disable_lua then
18.     return
19. end
```

Gambar 4.1 Konfigurasi pada init.lua

Jika file *scripting* berada di luar plug in (selain direktori “Wireshark/plugins/..”) dapat dimasukkan direktori ke dalam *scripting* pada Gambar 4.2, baris 700 digunakan untuk memasukkan direktori berada di posisi baris terakhir pada *scripting*.

```
20. if not running_superuser or run_user_scripts_when_superuser then
21. dofile(DATA_DIR.."scripting.lua")
22. end
```

Gambar 4.2 Konfigurasi file direktori pada init.lua

Langkah berikutnya adalah Membuat protokol baru, yang berisi beberapa variabel utama dan fungsi – fungsi yang akan dijalankan pada Gambar 4.3.

```
23. local http_response_proto = Proto("http_resp", "Upstream response HTTP request")
24. local http_shortcut_detail = Proto("http_shortcut", "Shortcut HTTP data")
25.
26. local stream_map = {}  -- table per stream : { req_cnt, resp_cnt, reqs }
27. local resp_map = {}
```

Gambar 4.3 Membuat protokol baru dalam *scripting*

Selanjutnya, membuat *dissector function* untuk TCP Sequence Analyzer (TSA) yang akan digunakan untuk membuat dan memanggil fungsi dijelaskan pada Gambar 4.4.

```
28. function tsa_menu_func()
29.     -- tsa analyze function
30.     function tsa_do(ip1, ip2)
31.        local results = {
32.           -- Total
33.           ["counter"] = 0,
34.           ["dupack"] = 0,
35.           ["retrans"] = 0,
36.           ["fastretrans"] = 0,
37.           ["zerowindow"] = 0,
38.           ["windowfull"] = 0,
39.
40.           -- dari ip1 ke ip2
41.           ["counter1to2"] = 0,
42.           ["dupack1to2"] = 0,
43.           ["retrans1to2"] = 0,
44.           ["fastretrans1to2"] = 0,
45.           ["zerowindow1to2"] = 0,
46.           ["windowfull1to2"] = 0,
47.
48.           -- dari ip2 ke ip1
49.           ["counter2to1"] = 0,
50.           ["dupack2to1"] = 0,
51.           ["retrans2to1"] = 0,
52.           ["fastretrans2to1"] = 0,
53.           ["zerowindow2to1"] = 0,
54.           ["windowfull2to1"] = 0,
55.        }
56.        local result_win = TextWindow.new("Tcp Sequence Numbers Analyze")
57.        local http_port = 80
```

Gambar 4.4 Membuat fungsi *dissector*

Setelah membuat dan memanggil fungsi, akan membuat tap. Tap adalah hasil langkah dari paket yang akan memproses hasil *capture* pada Gambar 4.5.

```
58. pemanggilan tap.draw
59.         function refresh_result()
60.            result_win:clear()
61.            result_win:set("Total\n")
62.            result_win:append("\tPackets counter: " .. results["counter"] .. "\n")
63.            result_win:append("\tDuplicate ACK: " .. results["dupack"] .. "\n")
64.            result_win:append("\tRetransmission: " .. results["retrans"] .. "\n")
65.            result_win:append("\tFast Retransmission: " .. results["fastretrans"] .. "\n")
66.            result_win:append("\tZero Window: " .. results["zerowindow"] .. "\n")
67.            result_win:append("\tWindow Full: " .. results["windowfull"] .. "\n")
68.            result_win:append("\n")
69.            result_win:append(ip1 .. " -> " .. ip2 .. "\n")
70.            result_win:append("\tPackets counter: " .. results["counter1to2"] .. "\n")
71.            result_win:append("\tDuplicate ACK: " .. results["dupack1to2"] .. "\n")
72.            result_win:append("\tRetransmission: " .. results["retrans1to2"] .. "\n")
73.            result_win:append("\tFast Retransmission: " .. results["fastretrans1to2"] .. "\n")
74.            result_win:append("\tZero Window: " .. results["zerowindow1to2"] .. "\n")
75.            result_win:append("\tWindow Full: " .. results["windowfull1to2"] .. "\n")
76.            result_win:append("\n")
77.            result_win:append(ip2 .. " -> " .. ip1 .. "\n")
78.            result_win:append("\tPackets counter: " .. results["counter2to1"] .. "\n")
79.            result_win:append("\tDuplicate ACK: " .. results["dupack2to1"] .. "\n")
80.            result_win:append("\tRetransmission: " .. results["retrans2to1"] .. "\n")
81.            result_win:append("\tFast Retransmission: " .. results["fastretrans2to1"] .. "\n")
82.            result_win:append("\tZero Window: " .. results["zerowindow2to1"] .. "\n")
83.            result_win:append("\tWindow Full: " .. results["windowfull2to1"] .. "\n")
84.         end
```

Gambar 4.5 Proses *output* pada *scripting* yang akan ditampilkan

Gambar 4.6 menjelaskan jumlah TSA paket akan ditampilkan setelah proses tap selesai, setiap baris mempunyai script yang sama, tetapi perbedaan hanya pemanggilan fungsi di setiap barisan, berikut adalah salah satu dari 18 fungsi *scripting* yang akan ditampilkan di window.

```
85. -- packets counter, total
86.         local counter_tap = Listener.new("frame", "ip.addr == " .. ip1 .. " && ip.addr == " .. ip2
    .. " && tcp.port == " .. http_port)
87.         function counter_tap.reset()
88.            results["counter"] = 0
89.         end
90.         function counter_tap.packet(pinfo, tvb, ip)
91.            results["counter"] = results["counter"] + 1
92.         end
93.         function counter_tap.draw()
```

```
94.            refresh_result()
95.         end
```

Gambar 4.6 *Scripting* untuk menampilkan jumlah seluruh paket

Untuk menghapus hasil jumlah paket tap pada baris 296 sampai baris 316 dijelaskan pada Gambar 4.7 menggunakan `remove_alltap`, fungsi ini untuk menutup aplikasi *plugin*

```
96.         function remove_alltap()
97.            counter_tap:remove()
98.            dupack_tap:remove()
99.            retrans_tap:remove()
100.           fastretrans_tap:remove()
101.           zerowindow_tap:remove()
102.           windowfull_tap:remove()
103.
104.           counter1to2_tap:remove()
105.           dupack1to2_tap:remove()
106.           retrans1to2_tap:remove()
107.           fastretrans1to2_tap:remove()
108.           zerowindow1to2_tap:remove()
109.           windowfull1to2_tap:remove()
110.
111.           counter2to1_tap:remove()
112.           dupack2to1_tap:remove()
113.           retrans2to1_tap:remove()
114.           fastretrans2to1_tap:remove()
115.           zerowindow2to1_tap:remove()
116.           windowfull2to1_tap:remove()
117.        End
118.
119.        result_win:set_atclose(remove_alltap)
120.
121.        retap_packets()
122.     end
```

Gambar 4.7 Menghilangkan dan menutup hasil TSA

Prompt alamat IP diperlukan untuk mengisi form berupa alamat IP. Pada Gambar 4.8, *scripting* ini juga untuk mengakhiri program TSA pada baris 327. Di baris 332, tree item adalah fungsi yang akan ditampilkan di detail paket Wireshark, *scripting* yang akan dibuat adalah respons dan jalan pintas untuk protokol HTTP.

```
123.    new_dialog("Please input the address pair", tsa_do, "ip address 1:", "ip address 2:")
124. End
```

```
125.
126.
127.
128.http_response_proto.fields.re_req_method = ProtoField.string("http_resp.request.method", "Request
    Method")
129.http_response_proto.fields.re_req_uri    = ProtoField.string("http_resp.request.uri", "Request
    URI")
130.http_response_proto.fields.re_req_ver    = ProtoField.string("http_resp.request.version",
    "Request Version")
131.http_response_proto.fields.re_host       = ProtoField.string("http_resp.host", "Host")
```

Gambar 4.8 *Scripting* untuk memasukkan Tree item

Kemudian di Gambar 4.9, baris 338 – 349 memberikan panggilan permintaan dan respons dalam protocol field untuk memasukkan proses fungsi kemudian.

```
132.http_shortcut_detail.fields.URL          = ProtoField.string("http.request.URL", "Request URL")
133.
134.local f_req_meth    = Field.new("http.request.method")
135.local f_req_uri     = Field.new("http.request.uri")
136.local f_req_ver     = Field.new("http.request.version")
137.local f_req_host    = Field.new("http.host")
138.local f_resp_code   = Field.new("http.response.code")
139.local f_tcp_stream  = Field.new("tcp.stream")
140.local f_tcp_dstport = Field.new("tcp.dstport")
141.local f_tcp_srcport = Field.new("tcp.srcport")
142.local f_ip_dsthost  = Field.new("ip.dst_host")
143.local f_ip_srchost  = Field.new("ip.src_host")
144.
145.function http_response_proto.init()
146.     stream_map = {}
147.     resp_map = {}
148.end
```

Gambar 4.9 Memasukkan untuk beberapa permintaan dan respons

Untuk fungsi TCP *port* pada Gambar 4.10 terbagi menjadi dua, yaitu baris 356 – 361 adalah optional port dan baris 363 – 368 adalah scheme by port. Masing – masing TCP *port* adalah port 443 dan port 80 di baris 357

```
149.local function optional_port(tcp_port)
150.     if tcp_port ~= 443 and tcp_port ~= 80 then
151.         return ":" .. tcp_port
152.     end
153.     return ""
154. end
```

```
155.
156. local function scheme_by_port(tcp_port)
157.     if tcp_port == 443 then
158.         return "https://"
159.     else
160.         return "http://"
161.     end
162. end
```

Gambar 4.10 Fungsi optional port dan scheme by port untuk TCP port.

Membuat fungsi protokol dissector dimulai dari baris ke-371 pada Gambar 4.11, yang berfungsi untuk memproses respons dan permintaan dalam satu fungsi. Permintaan dan respons memiliki fungsi yang berbeda, permintaan berfungsi ketika paket stream datang dan melakukan dissector paket. Sedangkan respons berfungsi jika ada paket stream yang datang mendapat respons, lalu mengarahkan secara arus balik dan mengarah ke permintaan yang sesuai.

```
163.function http_response_proto.dissector(tvbuffer, pinfo, treeitem)
164.       if not f_tcp_stream() then return end
165.       local stream_n
166.       local URL
167.       stream_n = f_tcp_stream().value
168.       URL = nil
169.
170.       if f_req_meth() then
171.           tcp_port = f_tcp_dstport().value
172.           local host
173.           if f_req_host() then
174.               host = f_req_host().value
175.           else
176.               host = f_ip_dsthost().value
177.           end
178.           URL = scheme_by_port (tcp_port) .. host .. optional_port(tcp_port) ..
    f_req_uri().value
179.           if not pinfo.visited then
180.               if stream_map[stream_n] == nil then
181.                   stream_map[stream_n] = {0,0,{}}
182.               end
```

Gambar 4.11 Memasukkan fungsi permintaan

Sedangkan di baris 392 sampai 394, perhitungan respons berfungsi jika ada paket stream yang datang mendapat respons, lalu baris 399 sampai dengan 412 mengarahkan arus balik, dan kembali ke permintaan yang sesuai Gambar 4.12.

```
183.                request_n = stream_map[stream_n][1] + 1
184.                stream_map[stream_n][1] = request_n
185.                stream_map[stream_n][3][ request_n ] = {f_req_meth().value, f_req_uri().value,
    f_req_ver().value, f_req_host().value}
186.            end
187.        End
188.
189.
190.        if f_resp_code() then
191.            local response_n
192.            if not pinfo.visited then
193.                if not stream_map[stream_n] then
194.                    warn("HTTP response not in stream (" .. pinfo.number .. ")")
195.                    response_n = 0
196.                else
197.                    response_n = stream_map[stream_n][2] + 1
198.                    stream_map[stream_n][2] = response_n
199.                    resp_map[pinfo.number] = response_n
200.                end
201.            else
202.                response_n = resp_map[pinfo.number]
203.            End
```

Gambar 4.12 Memasukkan fungsi arus balik respons ke permintaan.

Untuk fungsi TCP port dilihat pada Gambar 4.13 menggunakan fungsi percabangan umtuk menentukan hasil data lost di baris 431.

```
204.            tcp_port = f_tcp_srcport().value
205.            if response_n > 0 then
206.                local subtree = treeitem:add(http_response_proto, nil)
207.                local data = stream_map[stream_n][3][response_n]
208.                if data then
209.                    subtree:add(http_response_proto.fields.re_req_method, data[1]):set_generated()
210.                    subtree:add(http_response_proto.fields.re_req_uri, data[2]):set_generated()
211.                    subtree:add(http_response_proto.fields.re_req_ver, data[3]):set_generated()
212.                    if (data[4]) then
213.                        subtree:add(http_response_proto.fields.re_host, data[4]):set_generated()
214.                    end
215.                    host = data[4]
216.                    if not host then
217.                        host = f_ip_srchost().value
218.                    end
219.                    URL = scheme_by_port(tcp_port) .. host .. optional_port(tcp_port) .. data[2]
```

```
220.                Else
221.                    warn("HTTP request data lost (" .. stream_n .. "," .. response_n .. ")")
222.                end
223.            end
224.        end
```

Gambar 4.13 Fungsi respons untuk packet stream.

Post-dissector digunakan untuk menampilkan eksekusi pada aplikasi perangkat wireshark, posisi *post-dissector* sebagai akhir dari program *scripting* Lua dilihat pada Gambar 4.14.

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

Gambar 4.14 Fungsi register post-dissector.

4.2 Tahapan Menjalankan Dissector

4.2.1 Pengambilan Data

Pengambilan dilaksanakan pada tanggal 27 Agustus 2018, di lokasi Jalan Affandi, Soropadan, Condongcatur, Kecamatan Depok, Kabupaten Sleman, Daerah Istimewa Yogyakarta. Tabel 4.1 menjelaskan detail laporan pengambilan paket data yang dilakukan.

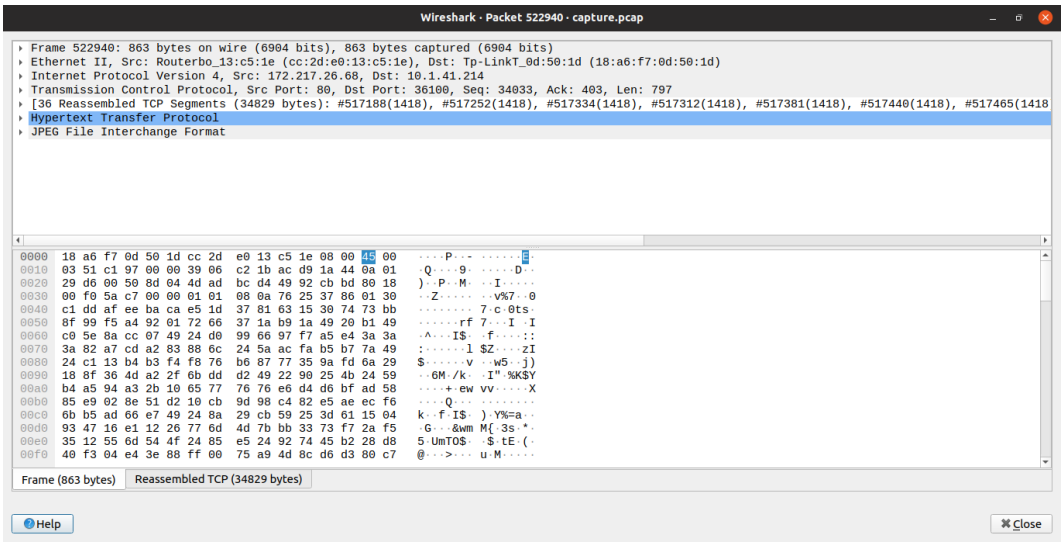
Tabel 4.1 Detail pengambilan paket data

Nama Tempat	Warunk Upnormal
Lokasi	Jalan Affandi, Soropadan, Condongcatur, Kecamatan Depok, Kabupaten Sleman, Daerah Istimewa Yogyakarta
Tanggal Pengambilan Data	27 Agustus 2018
Durasi Pengambilan Data	20:10 – 23:16 (3 jam)
Ukuran (Megabyte)	783 MB
Format	Wireshark Capture File (.pcap)
Nama File	Capture.pcap

4.2.2 Pengujian Pertama

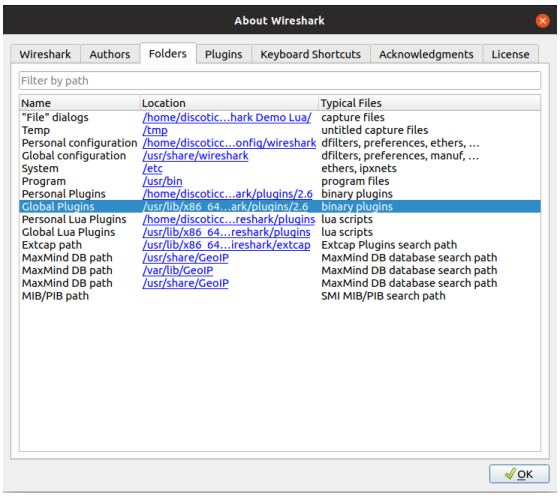
Pengujian *dissector* menunjukkan proses pembuatan *dissector* untuk protokol khusus yang beroperasi di protokol TCP dan HTTP, sebelum memiliki *dissector* protokol khusus,

Wireshark tidak mampu menafsirkan dan mengumpulkan beberapa paket protokol. Hasil *dissector* paket protokol khusus di Wireshark tanpa *add on* yang tepat ditunjukkan pada Gambar 4.15, dapat dilihat seluruh sub tree hanya bawaan aplikasi Wireshark saja.



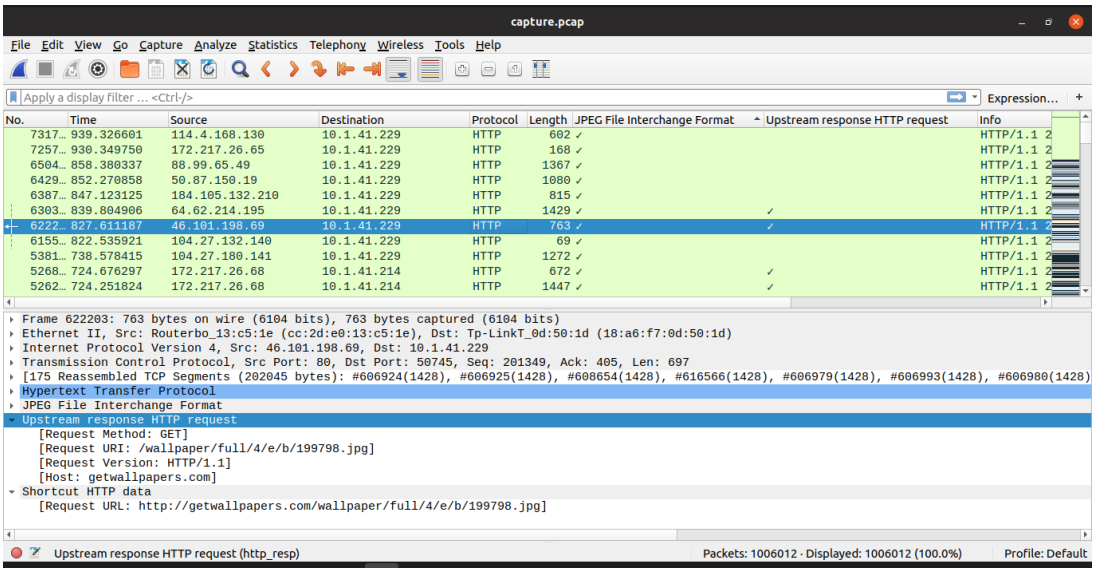
Gambar 4.15 Standar sub Tree Wireshark jika tidak menggunakan *scripting*.

Setelah implementasi pembuatan *scripting* selesai, kemudian pada Gambar 4.16 menjelaskan *scripting* akan dijalankan dengan dua perintah, dua perintah tersebut akan menyesuaikan tempat pada tampilan Wireshark yang akan dijalankan. Sebelum menjalankan *scripting*, pengguna terlebih dahulu memeriksa folder plugin di dalam direktori Wireshark, atau membuka menu bar About Wireshark (Help – About Wireshark), kemudian klik tab folders.



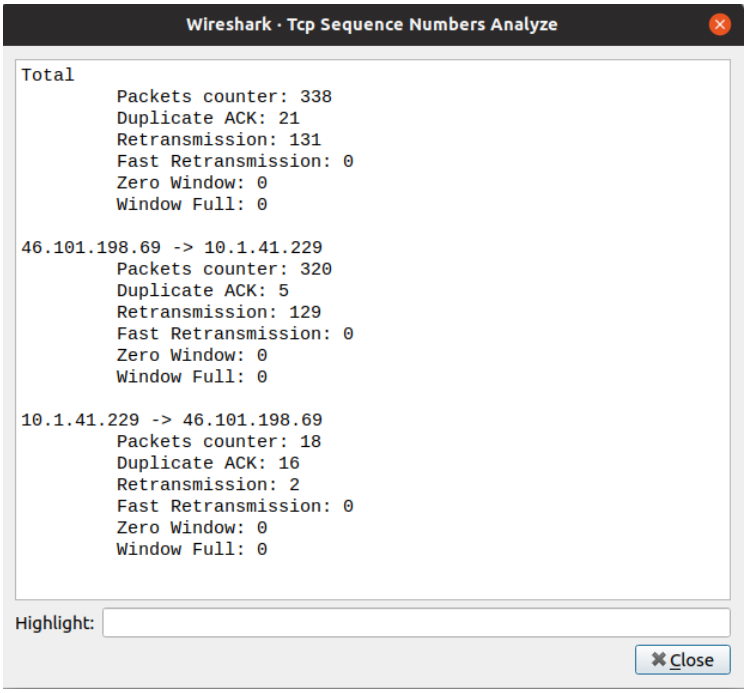
Gambar 4.16 Direktori Global Plugins untuk memasukkan dan modifikasi *scripting*.

Pengguna dapat memeriksa *scripting* yang telah dipasang pada tab plugins untuk memastikan munculnya *scripting*, jika tidak muncul gunakan kombinasi keyboard (ctrl+shift+L). Pada gambar 4.1, informasi yang digunakan untuk definisi protokol disajikan, baris pertama menunjukkan jenis informasi, kemudian baris kedua menunjukkan sumber URI (Uniform Resource Identifier), baris ketiga menunjukkan versi HTTP baris keempat menunjukkan URL (Uniform Resource Locator), serta baris kelima adalah gabungan dari URL dan URI. Menjalankan perintah program yang sebelumnya mengetahui salah satu paket IP sumber ke IP destinasi dan detail HTTP seperti pada gambar 4.1.



Gambar 4.17 Detail upstream respons pada pengujian pertama.

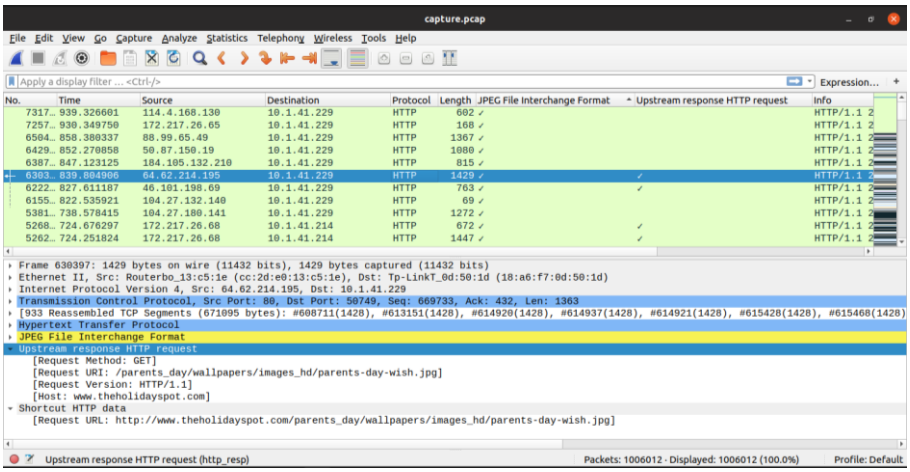
Setelah melakukan pencarian paket seperti di atas, TSA akan menghasilkan jumlah tap seperti pada Gambar 4.18, gambar tersebut hanya mendapatkan 3 detail, yaitu packet counter, duplicate ACK, dan retransmission. Hasil akhir pada TSA berbeda ketika IP source ke IP destinasi telah selesai dihitung, kemudian menghitung kembali secara arus balik dari IP destinasi ke IP source.



Gambar 4.18 Hasil *dissector* Wireshark pengujian pertama.

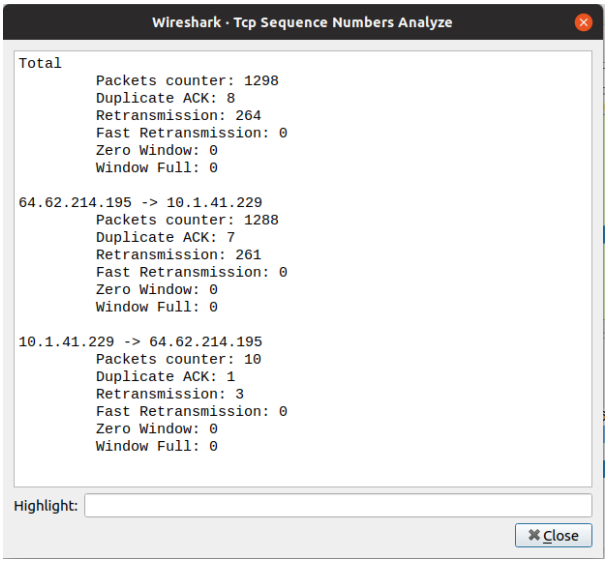
4.2.3 Pengujian Kedua

Pada Gambar 4.18, pengujian kedua menunjukkan respons HTTP yang berbeda, tetapi *IP* destinasi sama seperti pengujian pertama.



Gambar 4.19 Detail upstream respons pada pengujian kedua.

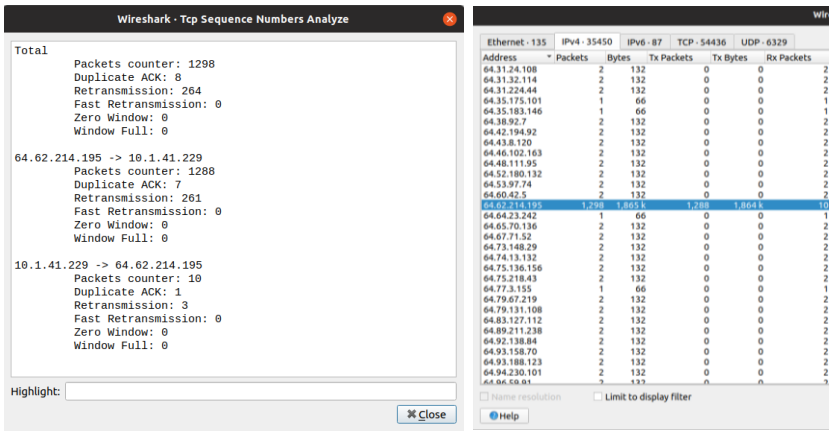
Hasil dari Gambar 4.20, tiga detail tap yang sama telah ditampilkan. Hasil penjumlahan berbeda dengan pengujian pertama, pengujian kedua mendapatkan tap lebih banyak daripada pengujian pertama.



Gambar 4.20 Hasil *dissector* Wireshark pengujian kedua.

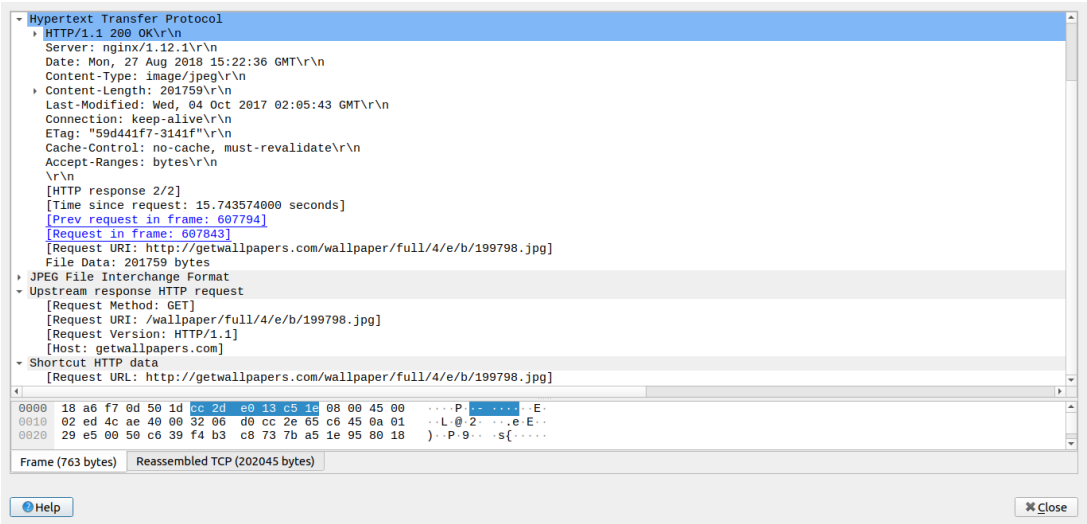
4.2.4 Hasil Perbandingan

Jumlah Hasil TSA dengan grafik statistik endpoint dari perhitungan telah ditampilkan, salah satu total paket pengujian mempunyai kemiripan yang sama. Pada Gambar 4.21 dapat dilihat persamaan dan perbedaan antara TSA dengan statistik endpoint, TSA lebih lengkap jika ditambahkan dengan 5 detail, sedangkan statistik endpoint hanya menghitung jumlah paket dengan satu arah dan harus mencari IP destinasi jika ingin menghitung jumlah paket untuk menyesuaikan, total seluruh paket dari TSA dan endpoint sebanyak 1298 paket. Jumlah paket data dalam 2 protokol yang berbeda sebanyak 1.298 paket, transfer paket sebanyak 1.288 paket, dan terima paket sebanyak 10 paket.



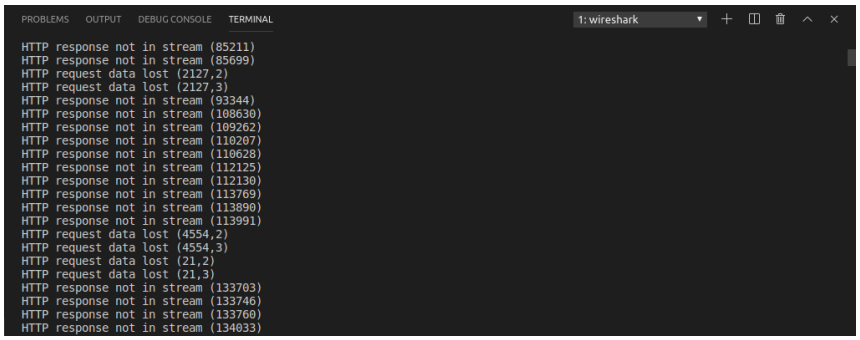
Gambar 4.21 Perbandingan TSA dengan grafik statistik endpoint.

Hasil berikutnya ialah perbandingan hasil output HTTP biasa dengan HTTP Request Response. Pada Gambar 4.22, menjelaskan antara kedua fungsi dengan hasil yang berbeda tergantung pada hasil paket yang telah ditampilkan. Perbandingan antara protokol HTTP Request tidak banyak memerlukan tempat atau lebih singkat dibandingkan dengan HTTP biasa yang memiliki banyak detail paket, serta posisi HTTP Request Response berada di bawah karena disebut sebagai protokol baru, terdiri dari request method, URI, Versi dan Host.



Gambar 4.22 Perbandingan protokol HTTP Script Lua dengan protokol HTTP default.

Bedasarkan gambar 4.23, Hasil output paket di *terminal* yang diproses melalui protokol HTTP Request Response, kadang-kadang mempunyai masalah seperti *error*, "HTTP Data not in stream" dan "Request data loss", biasanya hal ini disebabkan oleh paket yang terenkripsi, atau tidak *compatible*.



Gambar 4.23 Hasil Output HTTP Scripting Lua jika tidak menemukan protokol HTTP.