CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

To be able to access network services at a low level, it takes a module that has been provided by Python. The module is Socket. Here's a simple code for the packet sniffer by using the python built-in socket module:

```
1. import socket
2. def main():
3.       tangkap    =    socket.socket(    socket.AF_PACKET    ,
   socket.SOCK_RAW , socket.ntohs(0x0003))
4.       while True:
5.             packet = tangkap.recv(65535)
6.             print(packet)
7. main()
```

On line 1 of the program code contains the command to import the socket module, Line 2 is a command to create a **main()** function that will get the first process invoked. The function used is **socket.socket()** with three parameters inside. Here is the description of the parameters :

- **socket.AF_PACKET** is part of the Adreess Family supporting representations for low level packet interfaces.
- **socket.SOCK_RAW** is socket type used to read packet header content such as Ethernet Frame, IP Header, TCP Header and UDP Header.
- **socket.ntohs (0x0003)** is protocol with value 0x0003 in hexa, useful for receiving all Ethernet protocols

In line 4 of the program code contains the infinite loop command to run the code in rows 5 and 6. The **recv()** function in line 5 is uses to receive data packet from a connected socket with the buffer size is 65535 (Maximum value of the 16-bit total length field in the IP header). Line 6 to print the contents of captured data packets.

The result of the program code above, the data packet shown is still binary data. Therefore the data packet needs to be decoded. Decode starts from Ethernet Frame. Here's the function to decode Ethernet Frame:

```
8. def EthernetHeader(packet):
9.       frame = {}
10.      frame["mac_tujuan"] = mac_address(packet[0:6])
11.      frame["mac_sumber"] = mac_address(packet[6:12])
12.      frame["ether_type"] = ("%.2x"%(unpack('!H', packet[12:14])))
13.      frame["payload"] = packet[14:len(packet)]
14.      return frame
```

In line 8 the above program code contains a function called **EthernetHeader**. This function carries the value of packet data which is container in variable named **packet**. Lines 9 through 14 are block rows for the special function in them. Line 9 is a variable **frame** declaration as an array of Dictonary types. In Python, Dictonary is an array that can use numbers and strings as keys at each value. Ethernet Frame consists of 6 Byte Destination MAC addres, 6 Byte Source MAC address, 2 Byte Ethertype and 46 - 1500 Bytes payload. On lines 10 and 11 are variable declarations for add destination MAC value and source MACs into an **frame** array. Line 12 to add the protocol type value to the array **frame**. The unpack() function on line 12 is used for handling binary data. Line 13 is the declaration for the contents of the next header value with the key in the array **frame** is "**payload"**. Then all the frame data is returned on line 14.

The next header to be decoded is IPv4 Header. Here's the function to decode it :

```
15. def IPv4_Header(payload):
16.      packet = unpack("!BBHHHBBH4s4s",payload[:20])
17.      IPv4 = {}
18.      IPv4["version"] = int(bin(packet[0])[2:5],2)
19.      IPv4["ihl"] = int(bin(packet[0])[5:9],2)
20.      IPv4["TOS"] = int(bin(packet[1]),2)
21.      IPv4["total_length"] = packet[2]
22.      IPv4["identification"] = packet[3]
23.      p_4 = bin(packet[4])[2:].zfill(16)
24.      IPv4["DF"] = p_4[1:2]
25.      IPv4["DM"] = p_4[2:3]
26.      IPv4["Fragment_Offset"] = int(p_4[3:13],2)
```

```
27.      IPv4["ttl"] = packet[5]
28.      IPv4["protocol"] = packet[6]
29.      IPv4["checksum"] = packet[7]
30.      IPv4["ip_sumber"] = socket.inet_ntoa(packet[8]);
31.      IPv4["ip_tujuan"] = socket.inet_ntoa(packet[9]);
32.      IPv4["data"] = payload[20:len(payload)]
33.      return IPv4
```

IPv4 header length is 20 bytes. The core of the program code is on line 16 for the **packet** variable declaration. The value of the packet variable is the size of the IPv4 byte header that is divided into several parts by using the unpack function. Unpack function is useful for reads things from binary. The format '!' Is a byte order for network (= big-endian), 'B' is an integer type of data with a standard size of 1 Byte, 'H' is an integer data type with standard size 2 Byte, '4S' is a string data type with 4 Byte size. On lines 18 through 32 is a process for initializing fields in the IPv4 Header and returning the results in line 33. The **socket.inet_ntoa()** function on lines 30 and 31 serves to convert a 32-bit IPv4 address to a dotted-quad string format (for example, 123456789 to '123.45.67.89').

The next header is dependent on the resulting protocol. If the protocol value equals 6, then the next header refers to TCP Header. If the protocol value equal to 17, the next header refers to UDP Header. Here's the function to decode TCP Header :

```
34. def TCPHeader(payload):
35.      packet = unpack('!HHLLBBHHH' , payload[0:20]) # 2,2
  Byte
36.      segment = {}
37.      segment["port_sumber"] = packet[0]
38.      segment["port_tujuan"] = packet[1]
39.      segment["sequence"] = packet[2]
40.      segment["acknowledgement"] = packet[3]
41.      segment["header_length"]      =      (int(bin(packet[4])
  [2:6],2)*32)/8 # 4
42.      segment["reserved"] = bin(packet[4])[6:10] #
43.      segment["flag"] = packet[5]
44.      segment["window_size"] = packet[6]
45.      segment["checksum"] = packet[7]
46.      segment["urgent_pointer"] = packet[8]
47.      segment["data"] = ""
48.      for baca in payload[20:]:
49.            segment["data"] += (("%.2x " % (baca)))
```

```
50.      return segment
```

The **TCPHeader()** function will be automatically called if the protocol of the IPv4 field is 6. The rest of the packet that has not been decoded will be sent via the payload parameter. At line 35, the packet variable contains the value of the unpack result of the payload variable with an array length of 20 Bytes. On lines 36 to 49 is a process for initializing fields in the TCP Header and returning the results in line 50.

If the protocol value equals 17, then the next header refers to UDP Header. Here's the function to decode UDP Header :

```
51. def UDPHeader(payload):
52.      packet = unpack('!HHHH' , payload[0:8])
53.      datagram = {}
54.      datagram["port_sumber"] = packet[0]
55.      datagram["port_tujuan"] = packet[1]
56.      datagram["length"]  = packet[0]
57.      datagram["checksum"]  = packet[1]
58.      datagram["data"] = ""
59.      for baca in payload[8:]:
60.           datagram["data"] += (("%.2x " % (baca)))
61.      return datagram
```

All the above decode functions will be called in the infinity loop section of the main function to decode all captured packets.

5.2 Testing

There are two condition to test the Packet Sniffer program. The first condition where all computers are free from malware. The second condition where one computer is infected by malware. The malicious software used is malware that has activity on the network.

First Testing

In the first testing, all windows computers were not infected by malware. The picture below shows how the sniffer program was first run in terminal **PC Router.**

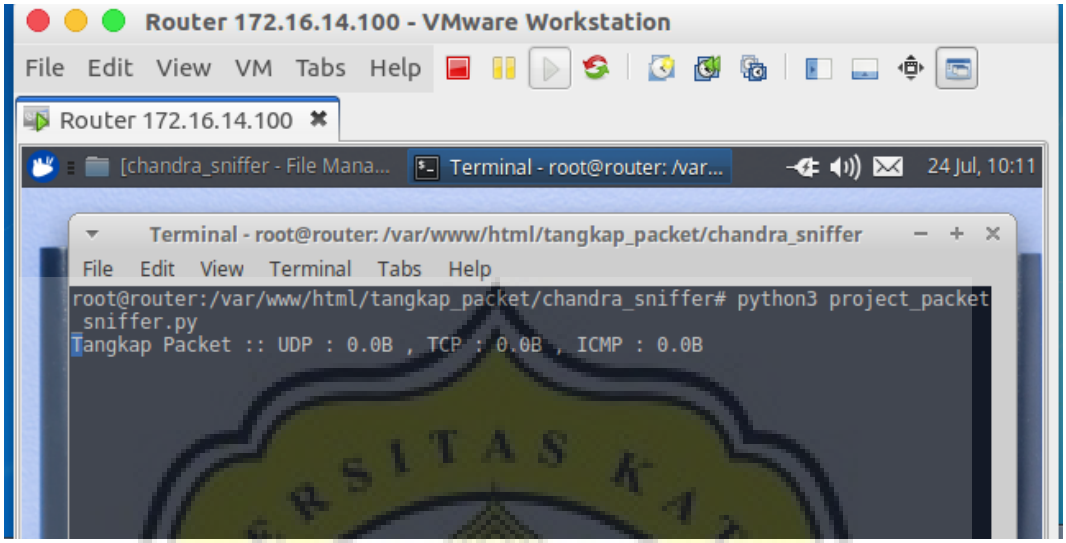

Illustration 5.1: Run the Packet Sniffer program

After the Sniffer packet is run, one of the computers access the internet. In this test **PC A** (192.168.15.2) accesses the youtube website, while **PC B** (192.168.15.3) does not access anything.

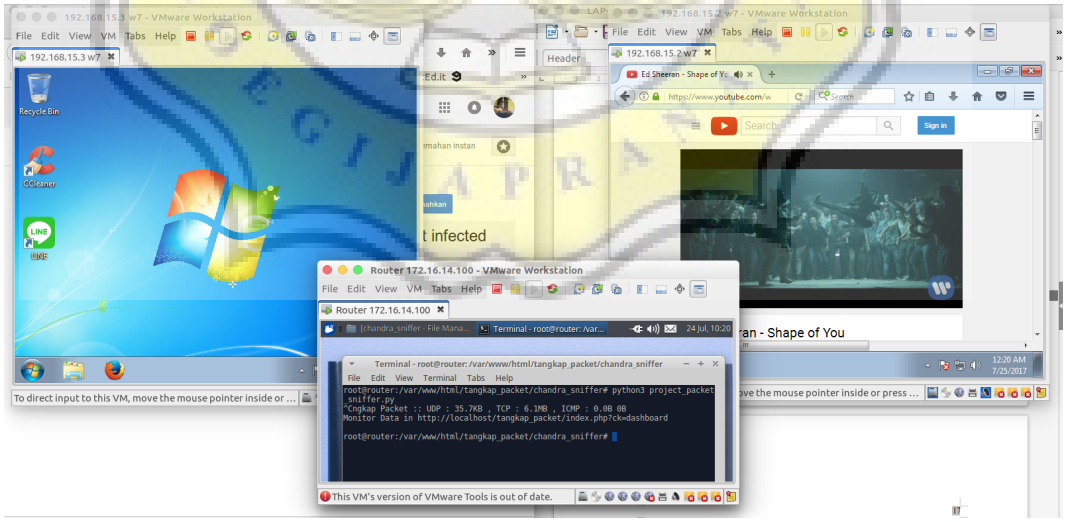

Illustration 5.2: PC Internet access and PC B did not access anything

Monitor packet capture results using a web-based program. Below is a dasboard page for monitoring captured results.

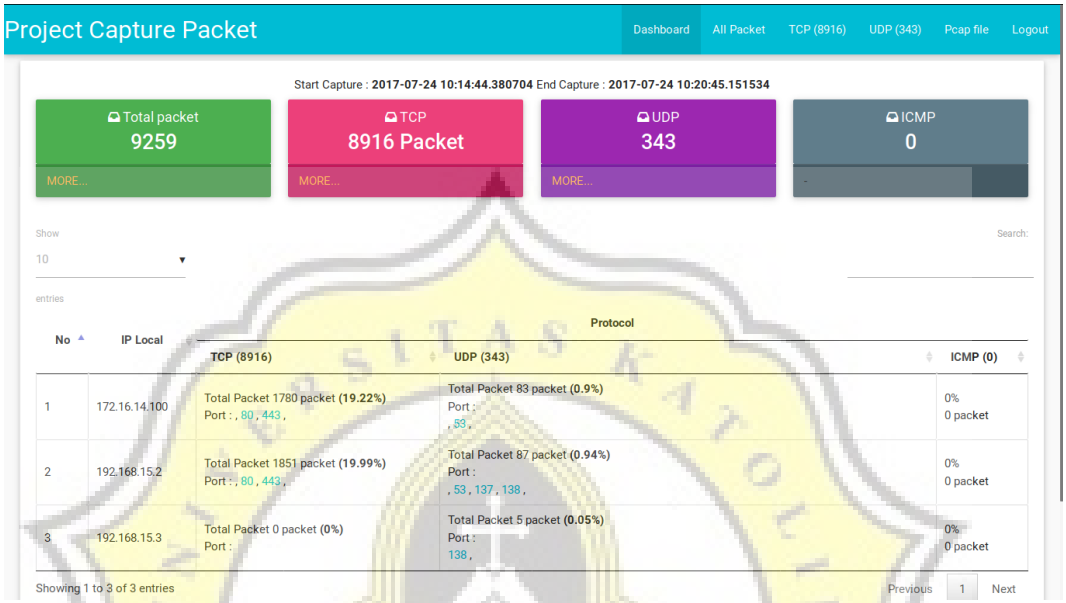

Illustration 5.3: Dashboard Page

Figure 5.3 above shows information that the program runs at 10:14 and stops at 10:28. **PC A** (192.168.15.2) has 1851 packets on TCP connections, accessing ports 80 and 443. **PC B** (192.168.15.3) there is no number of TCP packets because it does not access anything. Port 137, 138 on UDP is used for broadcast file sharing samba. Next to view all captured data packets.

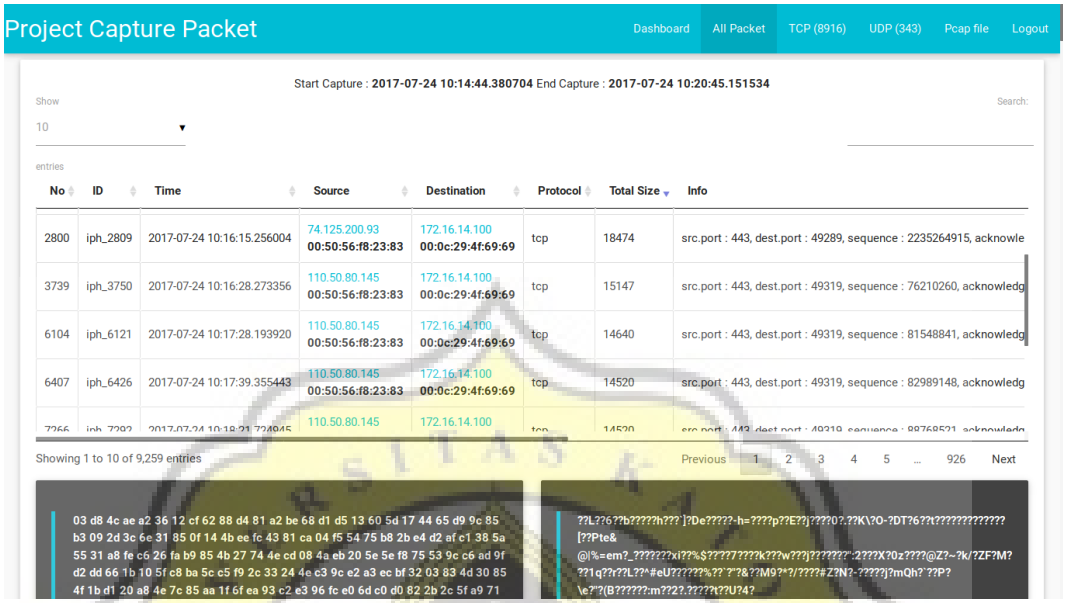

Illustration 5.4: Page to show all packets TCP and UDP

Figure 5.4 shows 1 to 10 of 9259 packets. The data displayed are TCP, UDP and ICMP protocols. The next page only shows the TCP data connection.

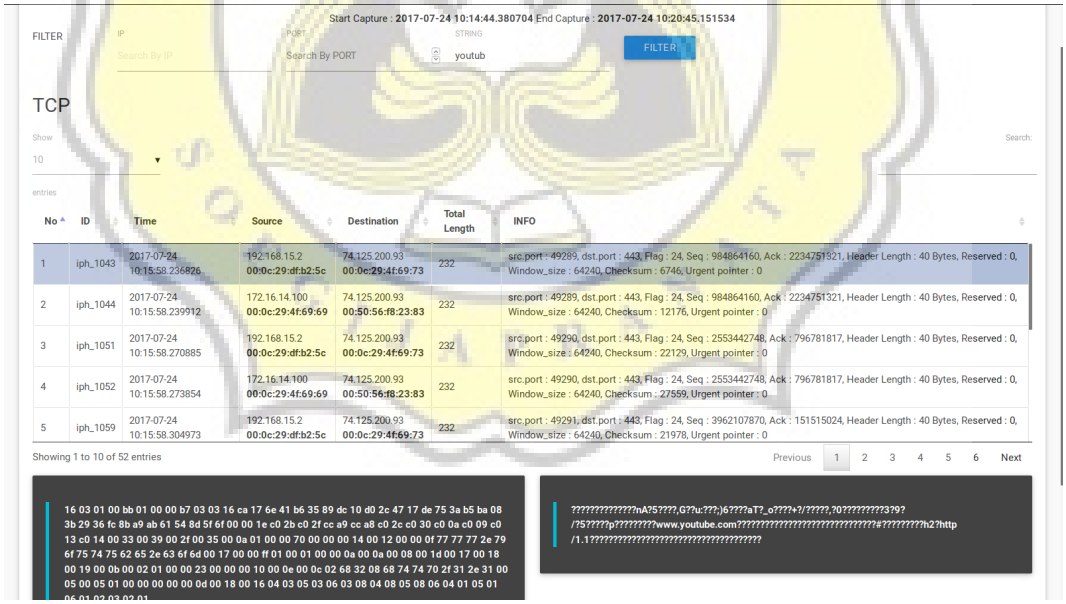

Illustration 5.5: Page to show packets TCP only

In the **Figure 5.5** above will show the data of TCP connection. When filtered with the keyword "youtube" the displayed data shows only the packets from **PC A** (192.168.15.2). The next page only shows the UDP data connection.

Illustration 5.6: Page to show packets UDP only

In **Figure 5.5** above shows the protocol UDP data. IP 8.8.8.8 is the IP of Google DNS accessed by **PC A** (192.168.15.2) and **PC Router** (172.16.14.100) via Port 53.

Second Testing

In the second test using ransomware type malware. This malware is named WannaCry. The **Figure 5.8** below shows the condition of **PC A** (192.168.15.3) that is infected by WannaCry malware and **PC B** (192.168.15.3) not yet infected.

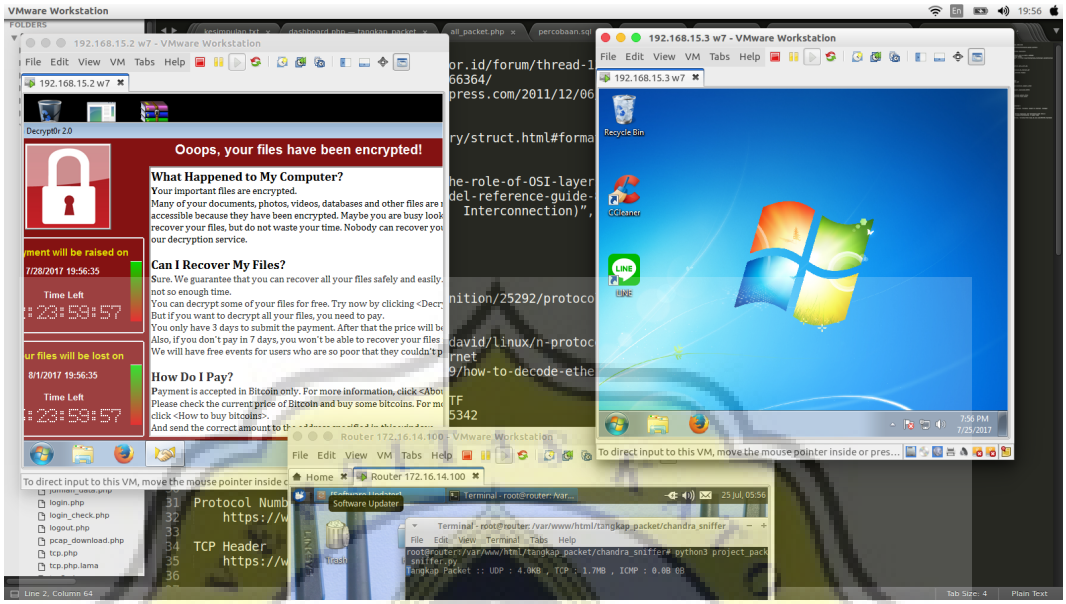

Illustration 5.7: PC A infected by WannaCry malware and PC B not yet infected.

In Figure above, **PC B** (192.168.15.3) has not been infected by malware. The **PC Router** (192.168.15.1) has run the Packet Sniffer program to record the data traffic from both Pcs. A minute later the **PC B** (192.168.15.3) computer gets infected. Time can be seen from the **PC B** (192.168.15.3) taskbar.

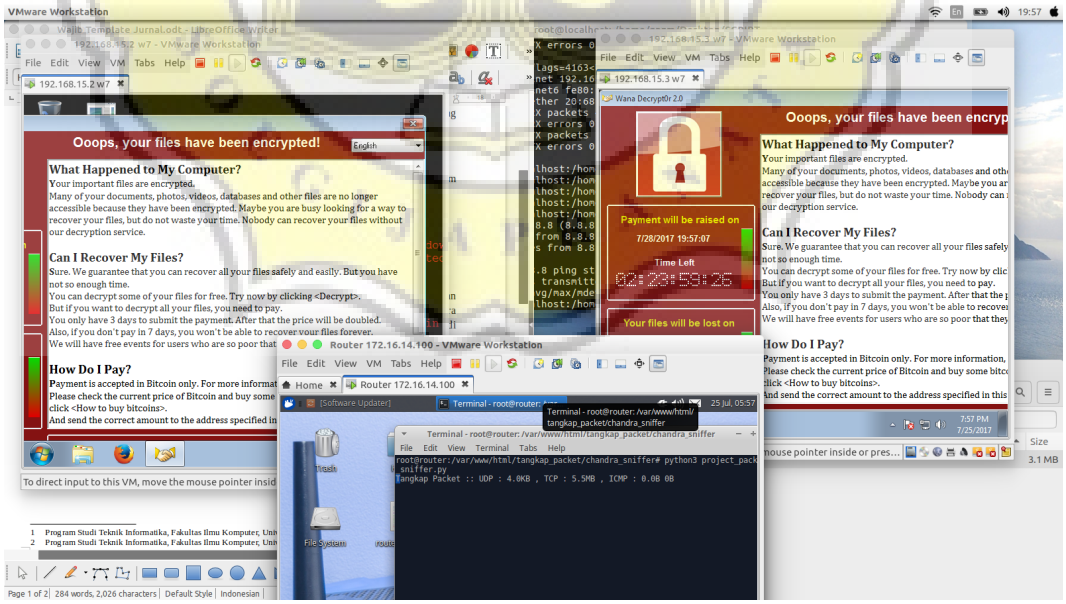

Illustration 5.8: PCB is infected by Malware

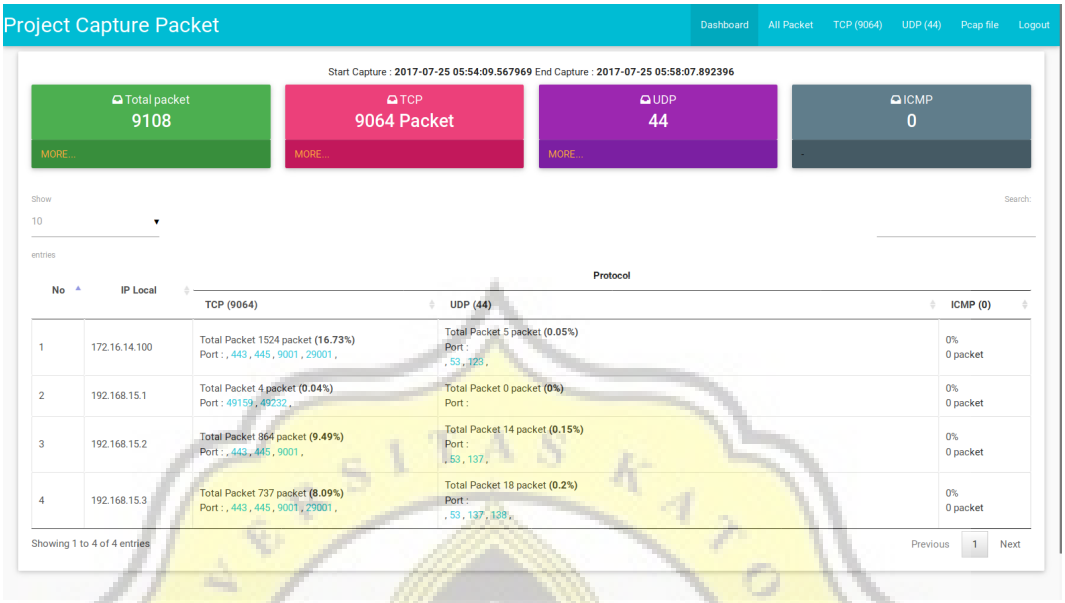

Illustration 5.9: Dashboard page after PCA and PCB is infected by malware

Based on **Figure 5.9** above, Shows information that the program runs at 05:54 and stops at 05:58. Within 6 minutes, **PC A** (192.168.15.2) sent 864 packets with TCP protocol and accessed through port 443,445,9001. **PC B** (192.168.15.3) sends packets from 737 packets with TCP protocol and access via port 443,445,9001,29001. **PC Router** (192.168.15.1) forwards the packets from **PC A** and **PC B** through a network interface that has an IP internet connection of 172.16.14.100. Port 137, 138 on UDP is used for samba sharing. port 53 is used for DNS servers and port 123 used for Network Time Protocol.

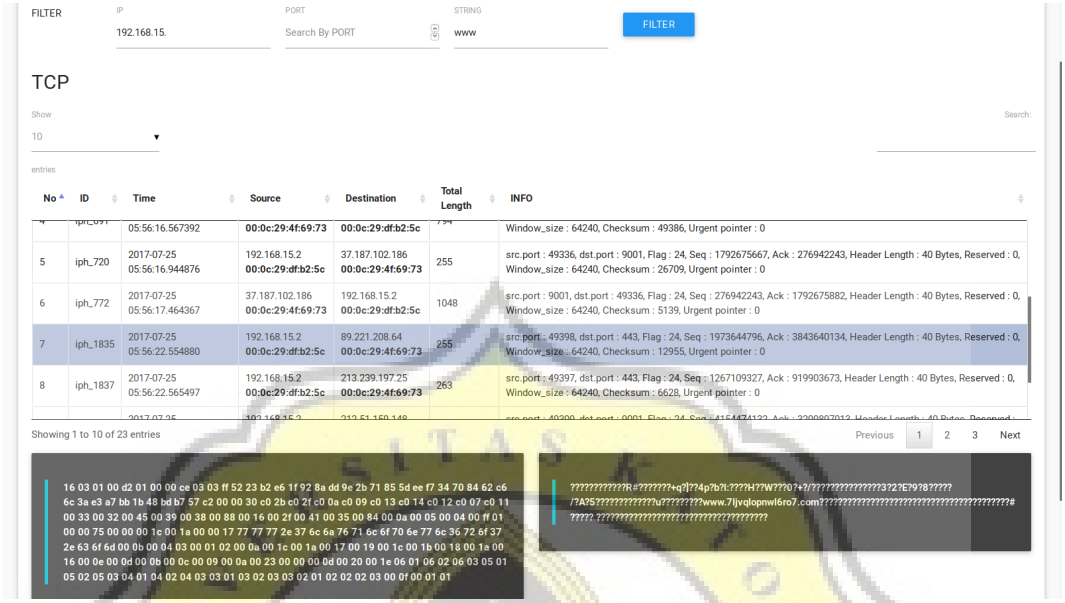

Illustration 5.10: TCP page after PCA and PCB is infected by malware

To view more specific data on the TCP protocol, **Figure 5.10** shows information from **PC A** (192.168.15.2) access to some Public IPs via ports 9001, 443 and 445 (if tables of all tables are displayed). Other than that there are data from each packet can be known.