

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 Address 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 Dictionary types. In Python, Dictionary is an array that can use numbers and strings as keys at each value. Ethernet Frame consists of 6 Byte Destination MAC address, 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('!HLLBBHHH' , 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"] += ("%0.2x " % (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**.

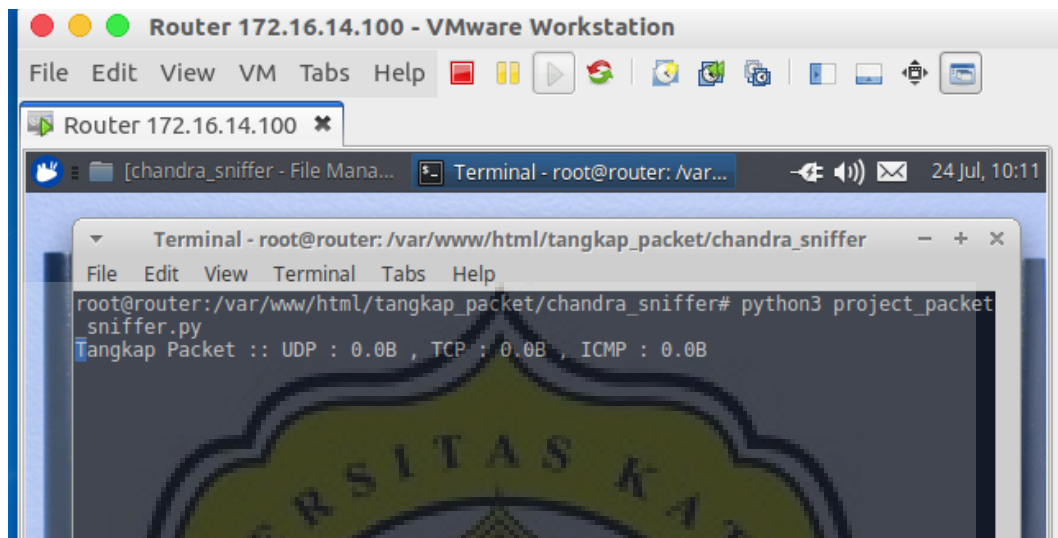


Illustration 5.1: Run the Packet Sniffer program

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.

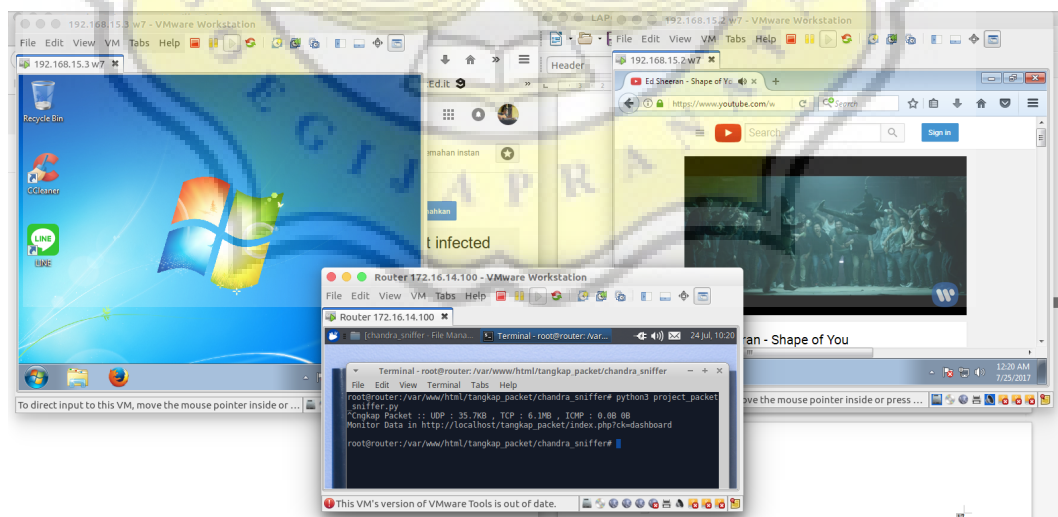


Illustration 5.2: PC Internet access and PC B did not access anything

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

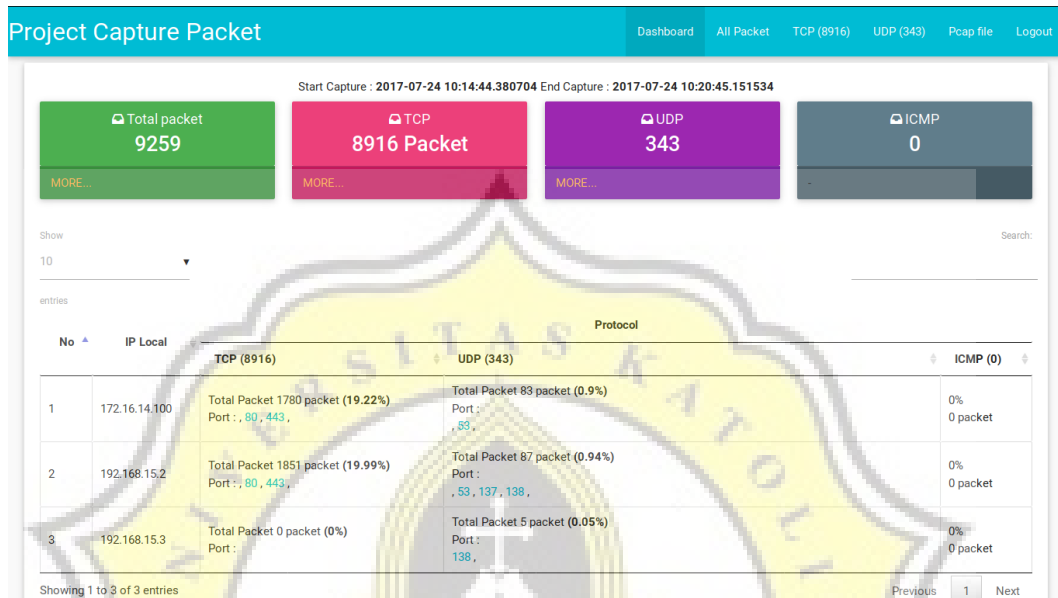


Illustration 5.3: Dashboard Page

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.

Project Capture Packet

Dashboard All Packet TCP (8916) UDP (343) Pcap file Logout

Start Capture : 2017-07-24 10:14:44.380704 End Capture : 2017-07-24 10:20:45.151534

Show: 10

entries

No	ID	Time	Source	Destination	Protocol	Total Size	Info
2800	iph_2809	2017-07-24 10:16:15.256004	74.125.200.93 00:50:56:f8:23:83	172.16.14.100 00:0c:29:4f:69:69	tcp	18474	src.port : 443, dest.port : 49289, sequence : 2235264915, acknowle
3739	iph_3750	2017-07-24 10:16:28.273356	110.50.80.145 00:50:56:f8:23:83	172.16.14.100 00:0c:29:4f:69:69	tcp	15147	src.port : 443, dest.port : 49319, sequence : 76210260, acknowledg
6104	iph_6121	2017-07-24 10:17:28.193920	110.50.80.145 00:50:56:f8:23:83	172.16.14.100 00:0c:29:4f:69:69	tcp	14640	src.port : 443, dest.port : 49319, sequence : 81548841, acknowledg
6407	iph_6426	2017-07-24 10:17:39.355443	110.50.80.145 00:50:56:f8:23:83	172.16.14.100 00:0c:29:4f:69:69	tcp	14520	src.port : 443, dest.port : 49319, sequence : 82989148, acknowledg
7966	iph_7983	2017-07-24 10:18:01.728045	110.50.80.145	172.16.14.100	tcp	14520	src.port : 443, dest.port : 49319, sequence : 88768571, acknowle

Showing 1 to 10 of 9,259 entries

Previous 1 2 3 4 5 ... 926 Next

03 d8 4c ae a2 36 12 cf 62 88 d4 81 a2 be 68 d1 d5 13 60 5d 17 44 65 d9 9c 85
b3 09 2d 3c 6e 31 85 0f 14 4b ee fc 43 81 ca 04 f5 54 75 b8 2b e4 d2 af e1 38 5a
55 31 a8 fe c6 26 fa b9 85 4b 27 74 4e cd 08 4a eb 20 5e 5e f8 75 53 9c c6 ad 9f
d2 dd 66 1b 10 5f e8 ba 5c e5 f9 2c 33 24 4e e3 9c e2 a3 ee bf 32 03 83 4d 30 85
4f 1b d1 20 a8 4e 7c 85 aa 1f 6f ea 93 c2 e3 96 fe e0 6d c0 d0 82 2b 2c 5f a9 71

??L?6??b????h???]De????-h????p?E?]?????.??K\?O-?DT?6??h?????????
[?Pte&
@!%=em?_?????x??\$?????????K??w??]??????..????X?o????@Z?-?K/?ZF?M?
??!q?h??L??*?eU??????%?? ?78??M9??*?/????Z?N?..?????mQh? ?P?
?e??(B??????m??Z?..?????h?U?4?

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.

Start Capture : 2017-07-24 10:14:44.380704 End Capture : 2017-07-24 10:20:45.151534

FILTER

IP Search By IP

PORT Search By PORT

STRING Search By STRING

Search By IP

Search By PORT

Search By STRING

youtube

FILTER

TCP

Show: 10

entries

No	ID	Time	Source	Destination	Total Length	INFO
1	iph_1043	2017-07-24 10:15:58.236826	192.168.15.2 00:0c:29:df:b2:5c	74.125.200.93 00:0c:29:4f:69:73	232	src.port : 49289, dst.port : 443, Flag : 24, Seq : 984864160, Ack : 2234751921, Header Length : 40 Bytes, Reserved : 0, Window_size : 64240, Checksum : 6746, Urgent pointer : 0
2	iph_1044	2017-07-24 10:15:58.239912	172.16.14.100 00:0c:29:4f:69:69	74.125.200.93 00:50:56:f8:23:83	232	src.port : 49289, dst.port : 443, Flag : 24, Seq : 984864160, Ack : 2234751921, Header Length : 40 Bytes, Reserved : 0, Window_size : 64240, Checksum : 12176, Urgent pointer : 0
3	iph_1051	2017-07-24 10:15:58.270885	192.168.15.2 00:0c:29:df:b2:5c	74.125.200.93 00:0c:29:4f:69:73	232	src.port : 49290, dst.port : 443, Flag : 24, Seq : 2553442748, Ack : 796781817, Header Length : 40 Bytes, Reserved : 0, Window_size : 64240, Checksum : 22129, Urgent pointer : 0
4	iph_1052	2017-07-24 10:15:58.273854	172.16.14.100 00:0c:29:4f:69:69	74.125.200.93 00:50:56:f8:23:83	232	src.port : 49290, dst.port : 443, Flag : 24, Seq : 2553442748, Ack : 796781817, Header Length : 40 Bytes, Reserved : 0, Window_size : 64240, Checksum : 27559, Urgent pointer : 0
5	iph_1059	2017-07-24 10:15:58.304973	192.168.15.2 00:0c:29:df:b2:5c	74.125.200.93 00:0c:29:4f:69:73	232	src.port : 49291, dst.port : 443, Flag : 24, Seq : 3962107870, Ack : 151515024, Header Length : 40 Bytes, Reserved : 0, Window_size : 64240, Checksum : 21978, Urgent pointer : 0

Showing 1 to 10 of 52 entries

Previous 1 2 3 4 5 6 Next

16 03 01 00 0b 01 00 00 07 03 03 16 ca 17 6e 41 b6 35 89 de 10 d0 2c 47 17 de 75 3a b5 ba 08
3b 29 36 fe 8b a9 ab 61 54 8d 5f 6f 00 00 1e c0 2b c0 2f cc a9 cc a8 c0 2c c0 30 c0 0a c0 09 c0
13 c0 14 00 33 00 39 00 2f 00 35 00 0a 01 00 00 70 00 00 00 14 00 12 00 00 01 77 77 77 2e 79
6f 75 74 75 62 65 2e 63 6f 6d 00 17 00 00 ff 01 00 01 00 00 0a 00 0a 00 08 00 14 00 17 00 18
00 19 00 0b 00 02 01 00 00 23 00 00 10 00 00 0c 02 68 32 08 68 74 74 70 21 31 2e 31 00
05 00 05 01 00 00 00 00 00 0d 00 18 00 16 04 03 05 03 06 03 08 04 08 05 08 06 04 01 05 01
06 01 02 03 03 03

?????????????hA5????G?h????6????hT_?s?????/?????h?????????399
/15????p????????www.youtube.com????????????????????????????????????h2http
/1.1??

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.

UDP

Show 10

Search:

entries

No	ID	Time	Source	Destination	Total Length	INFO
1	iph_6875	2017-07-24 10:18:05.742840	192.168.15.2 00:0c:29:d2:b2:5c	192.168.15.255 ff:ff:ff:ff:ff:ff	78	src.port : 137, dest.port : 137, Length : 137, Checksum : 137
4	iph_7834	2017-07-24 10:19:25.146335	8.8.8.8 00:50:56:f8:23:83	172.16.14.100 00:0c:29:4f:69:69	190	src.port : 53, dest.port : 61137, Length : 53, Checksum : 61137
5	iph_7835	2017-07-24 10:19:25.149567	8.8.8.8 00:0c:29:4f:69:73	192.168.15.2 00:0c:29:d2:b2:5c	190	src.port : 53, dest.port : 61137, Length : 53, Checksum : 61137
2	iph_7826	2017-07-24 10:19:25.103169	192.168.15.2 00:0c:29:d2:b2:5c	8.8.8.8 00:0c:29:4f:69:73	59	src.port : 61137, dest.port : 53, Length : 61137, Checksum : 53
3	iph_7827	2017-07-24 10:19:25.106177	172.16.14.100 00:0c:29:4f:69:69	8.8.8.8 00:50:56:f8:23:83	59	src.port : 61137, dest.port : 53, Length : 61137, Checksum : 53

Showing 1 to 5 of 5 entries

Previous 1 Next

f3 98 01 10 00 01 00 00 00 00 00 20 46 44 46 45 46 46 45 45 45 46 45 4f 46
45 45 43 43 41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 00 00 20 00 01

???????????? FDFEFFFFEEDFECCACACACACACACACA?? ??

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.

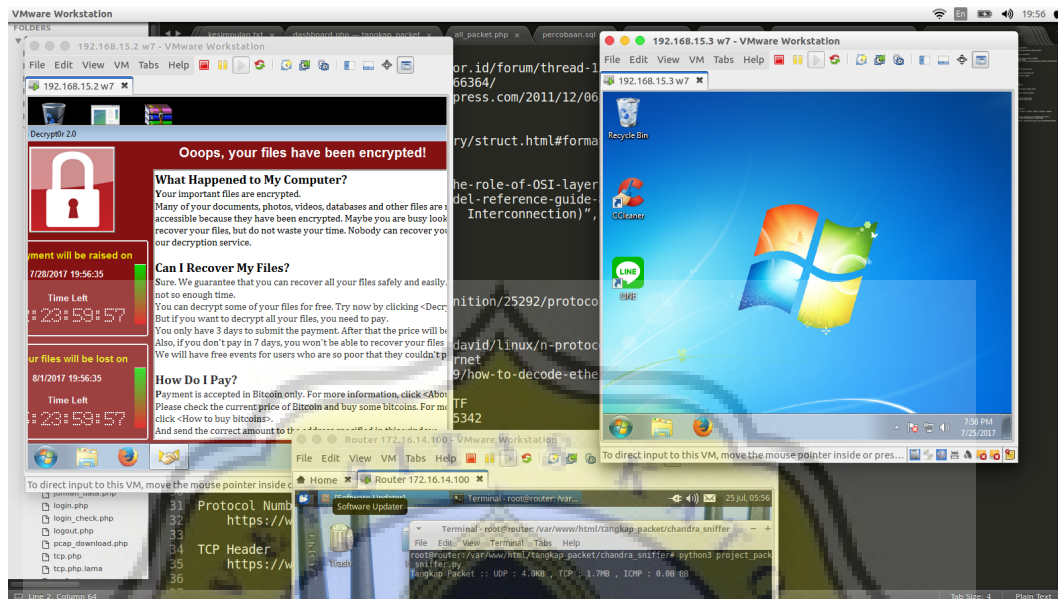


Illustration 5.7: PC A infected by WannaCry malware and PC B not yet infected.

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.

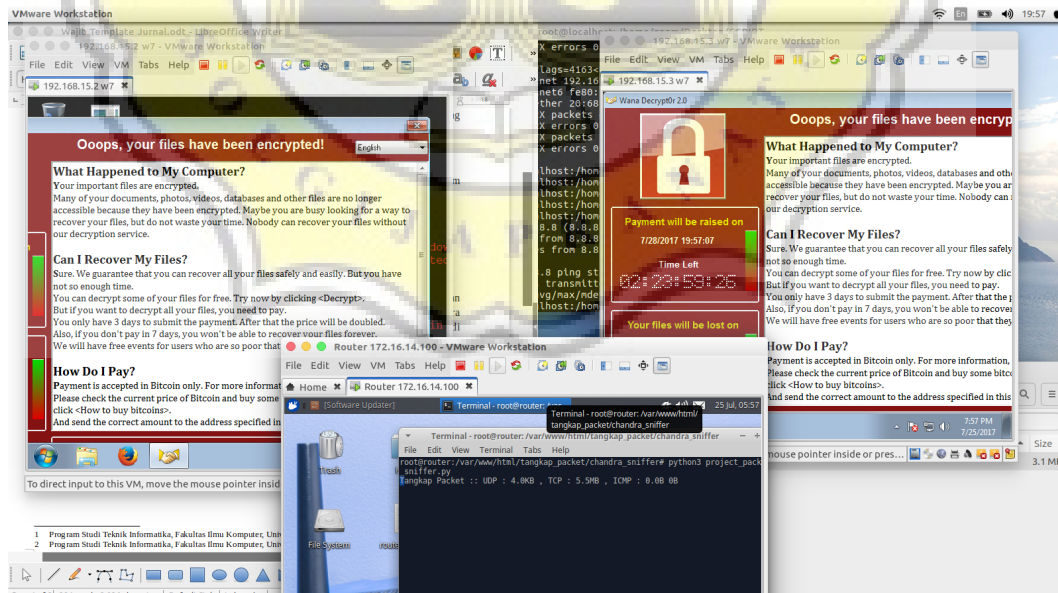


Illustration 5.8: PCB is infected by Malware

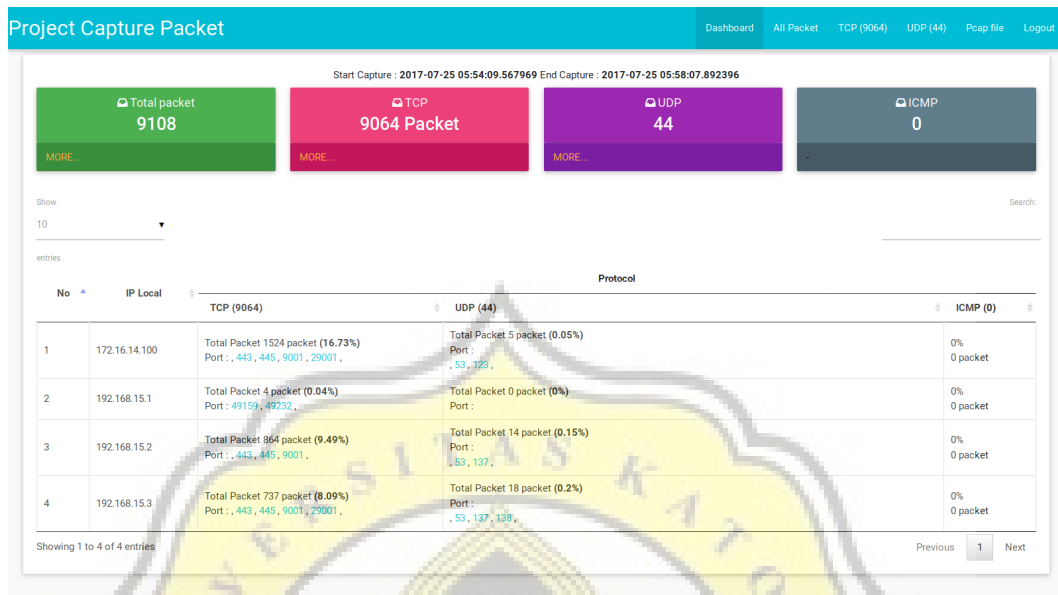


Illustration 5.9: Dashboard page after PCA and PCB is infected by malware

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.

FILTER

IP: 192.168.15.2 PORT: Search By PORT STRING: www FILTER

TCP

Show: 10

Search:

entries

No	ID	Time	Source	Destination	Total Length	INFO
5	iph_720	2017-07-25 05:56:16.944876	192.168.15.2	37.187.102.186	255	src.port: 49336, dst.port: 9001, Flag: 24, Seq: 1792675667, Ack: 276942243, Header Length: 40 Bytes, Reserved: 0, Window_size: 64240, Checksum: 26709, Urgent pointer: 0
6	iph_772	2017-07-25 05:56:17.464367	37.187.102.186	192.168.15.2	1048	src.port: 9001, dst.port: 49336, Flag: 24, Seq: 276942243, Ack: 1792675882, Header Length: 40 Bytes, Reserved: 0, Window_size: 64240, Checksum: 5139, Urgent pointer: 0
7	iph_1835	2017-07-25 05:56:22.554880	192.168.15.2	89.221.208.64	255	src.port: 49398, dst.port: 443, Flag: 24, Seq: 1973644796, Ack: 3843640134, Header Length: 40 Bytes, Reserved: 0, Window_size: 64240, Checksum: 12955, Urgent pointer: 0
8	iph_1837	2017-07-25 05:56:22.565497	192.168.15.2	213.239.197.25	263	src.port: 49397, dst.port: 443, Flag: 24, Seq: 1267109327, Ack: 919903673, Header Length: 40 Bytes, Reserved: 0, Window_size: 64240, Checksum: 6628, Urgent pointer: 0

Showing 1 to 10 of 23 entries

Previous 1 2 3 Next

```

16 03 01 00 d2 01 00 00 ce 03 03 ff 52 23 b2 e6 1f 92 8a dd 9e 2b 71 85 5d ee f7 34 70 84 62 c6
6c 3a e3 a7 bb 1b 48 bd b7 57 c2 00 00 30 c0 2b c0 21 c0 0a c0 09 c0 13 c0 14 c0 12 c0 07 c0 11
00 33 00 32 00 45 00 39 00 38 00 88 00 16 00 2f 00 41 00 35 00 84 00 0a 00 05 00 04 00 ff 01
00 00 75 00 00 00 1c 00 1a 00 00 17 77 77 77 2e 37 6c 6a 76 71 6c 6f 70 6e 77 6c 36 72 6f 37
2e 63 6f 6d 00 0b 00 04 03 00 01 02 00 0a 00 1c 00 1a 00 17 00 19 00 1c 00 1b 00 18 00 1a 00
16 00 0e 00 0d 00 0b 00 0c 00 09 00 0a 00 23 00 00 0d 00 20 00 1e 06 01 06 02 06 03 05 01
05 02 05 03 04 01 04 02 04 03 03 01 03 02 03 03 02 01 02 02 03 00 0f 00 01 01

```

```

?????????R?????+g?74p7b7:????H?W?????+7/?????????????32?E978????
7A95????????????????www.7ljvlopnl6ro7.com?????????????????????????????#
????????????????????????????????????????????????????????????????

```

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.