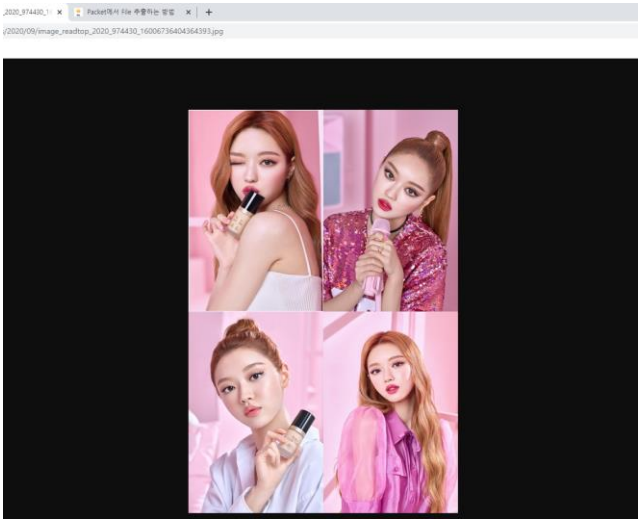
Assignment 1

2016147030 송민석

1-1.

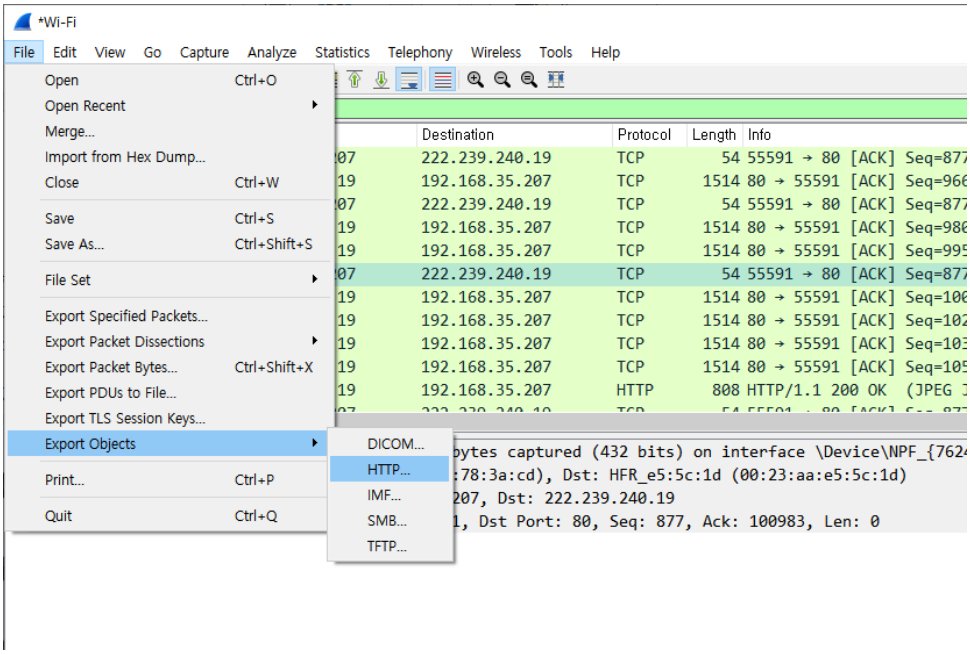
다음 Web page의 이미지를 패킷 분석을 통해 Extract하고자 한다.

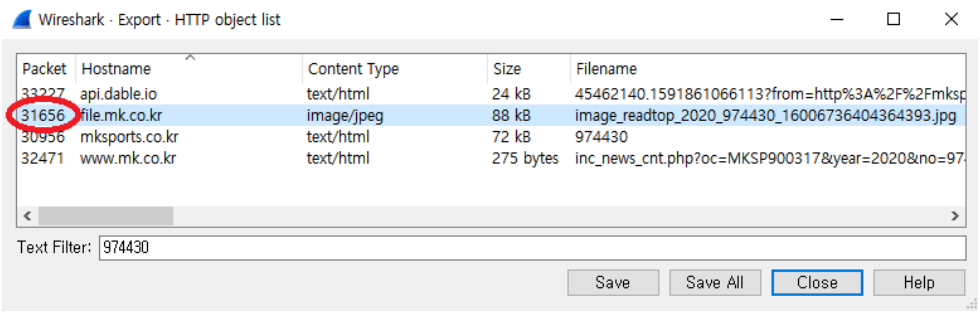
URL : http://file.mk.co.kr/meet/neds/2020/09/image_readtop_2020_974430_16006736404364393.jpg



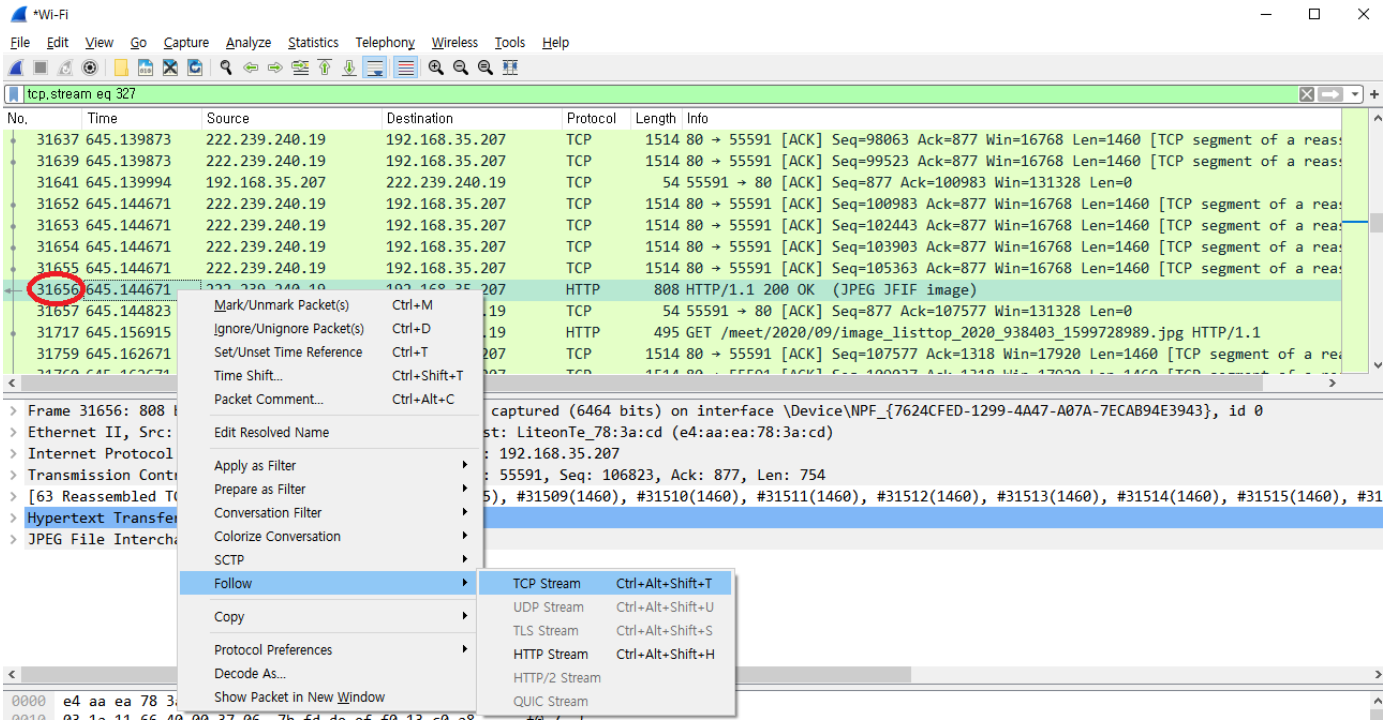
WiFi 네트워크를 이용하고 있었으므로, WireShark 를 실행한 후 WiFi 네트워크의 패킷 캡쳐를 시작한다. 이후 브라우저를 통해 해당 이미지의 URL 에 접속하여 Request / Response 를 Wireshark 가 기록하도록 한다.

Export Objects – HTTP 를 통해 해당 이미지의 HTTP Request/Response 의 Packet No. 를 확인한다.



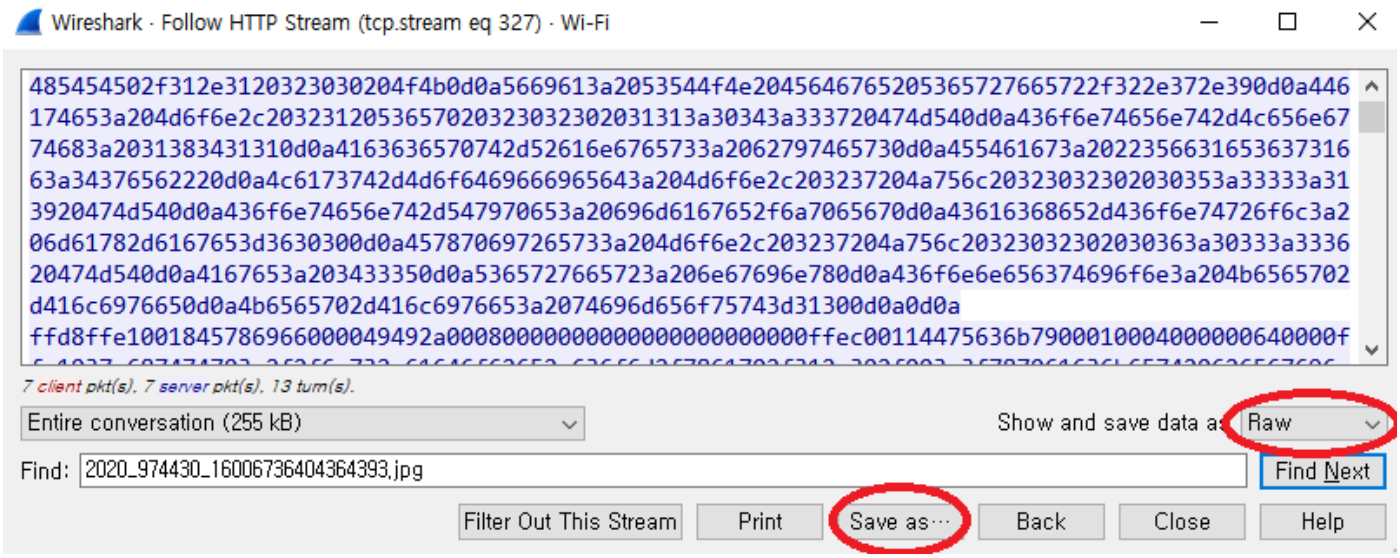


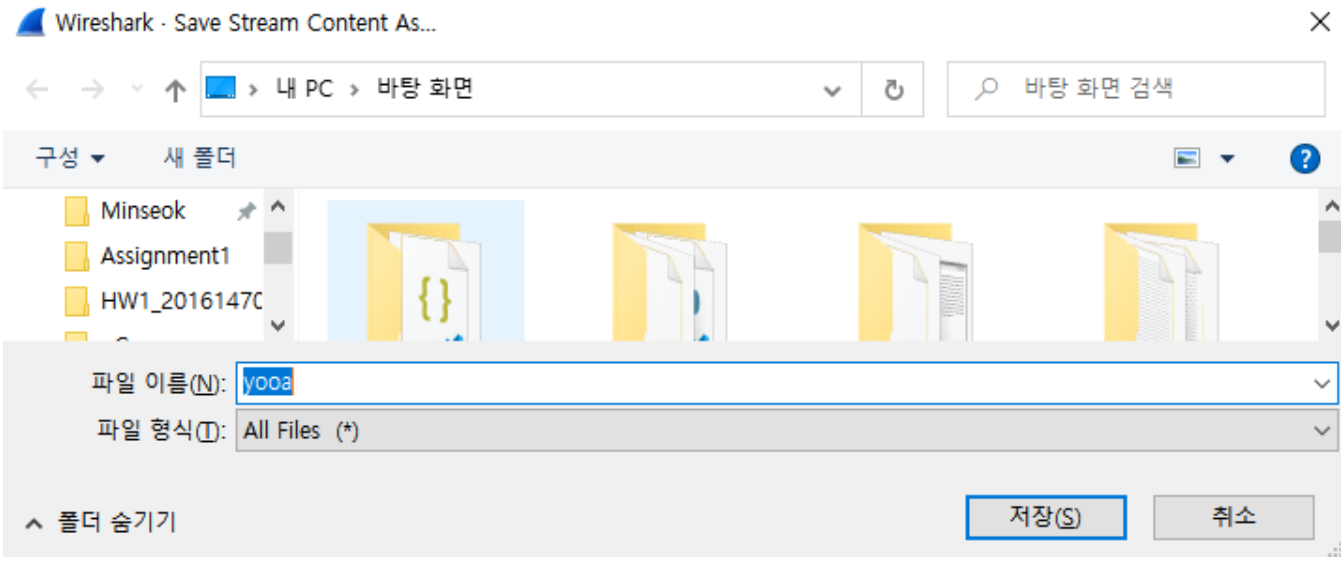
31656 번 패킷을 찾아 Response 의 내용을 추적한다.



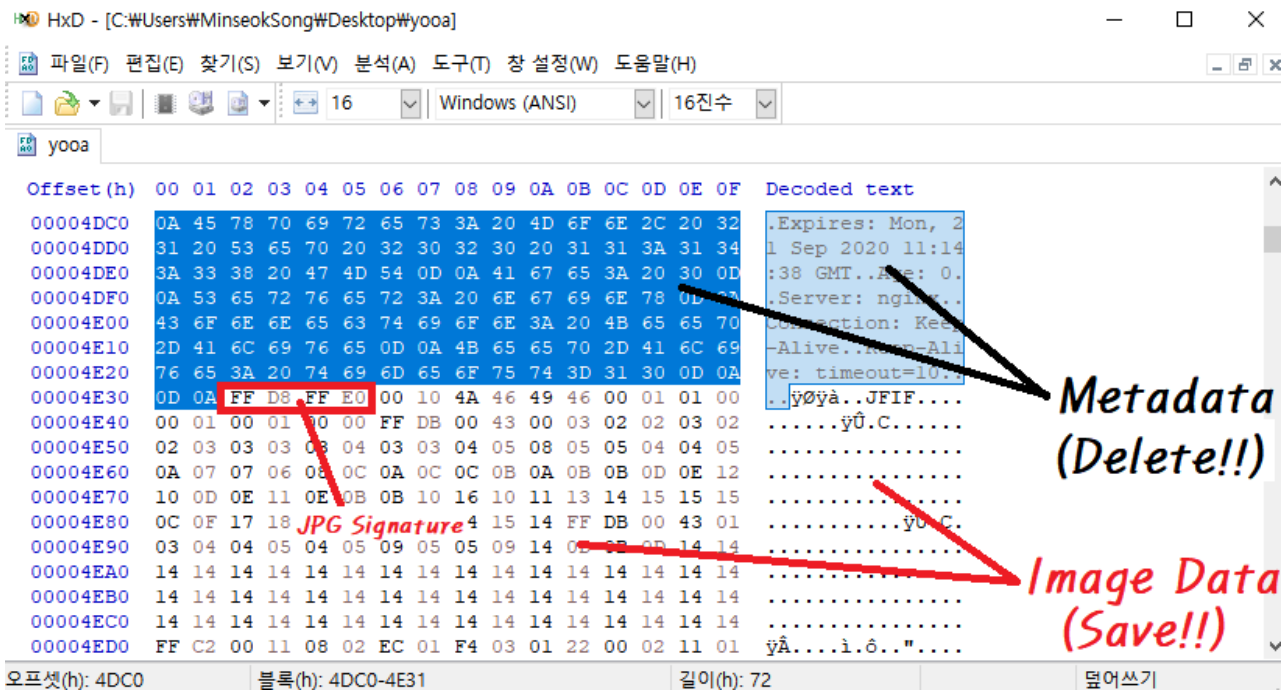
패킷 내부에서 해당 이미지의 Request / Response 를 찾아낸다.

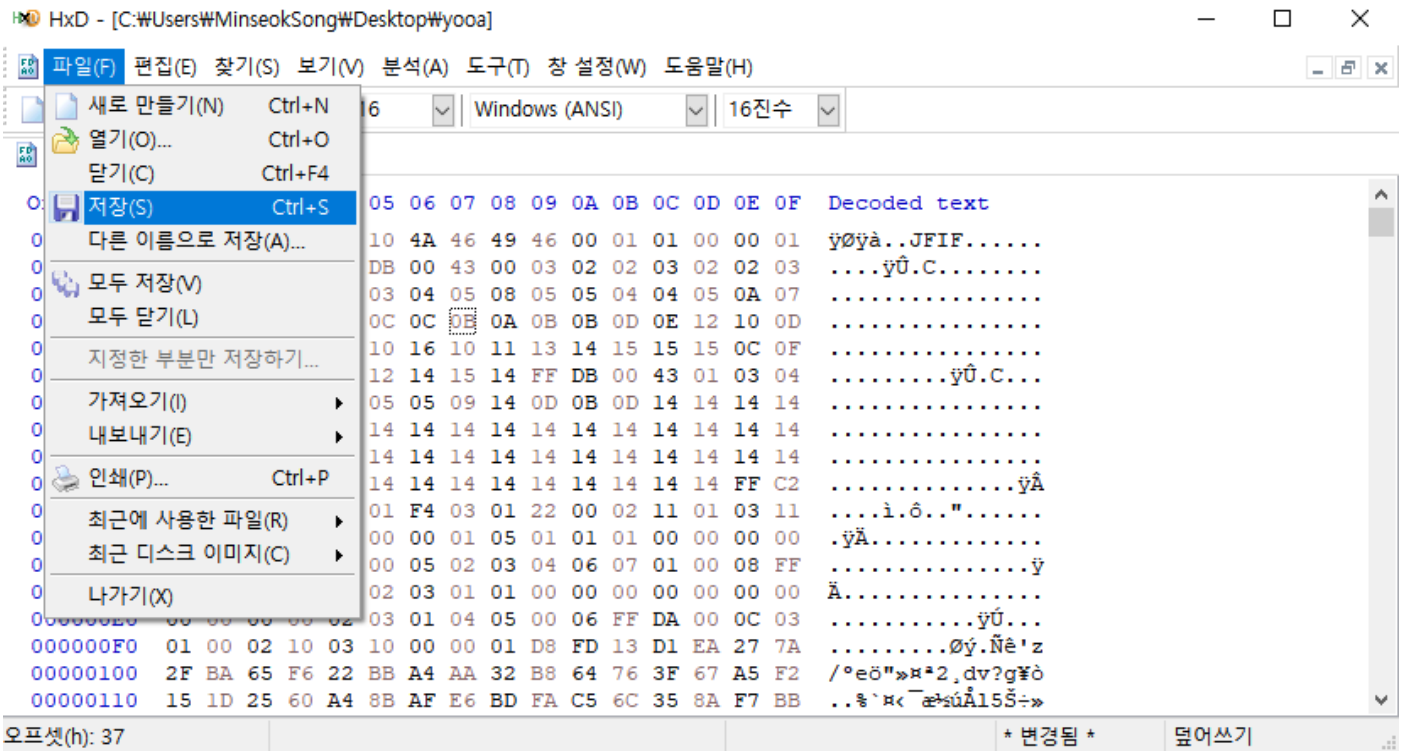
해당 내용이 ASCII Code 형태로 되어 있어 확인하기 힘드므로, 이를 Raw format(16 진수 형태)로 저장하였다.



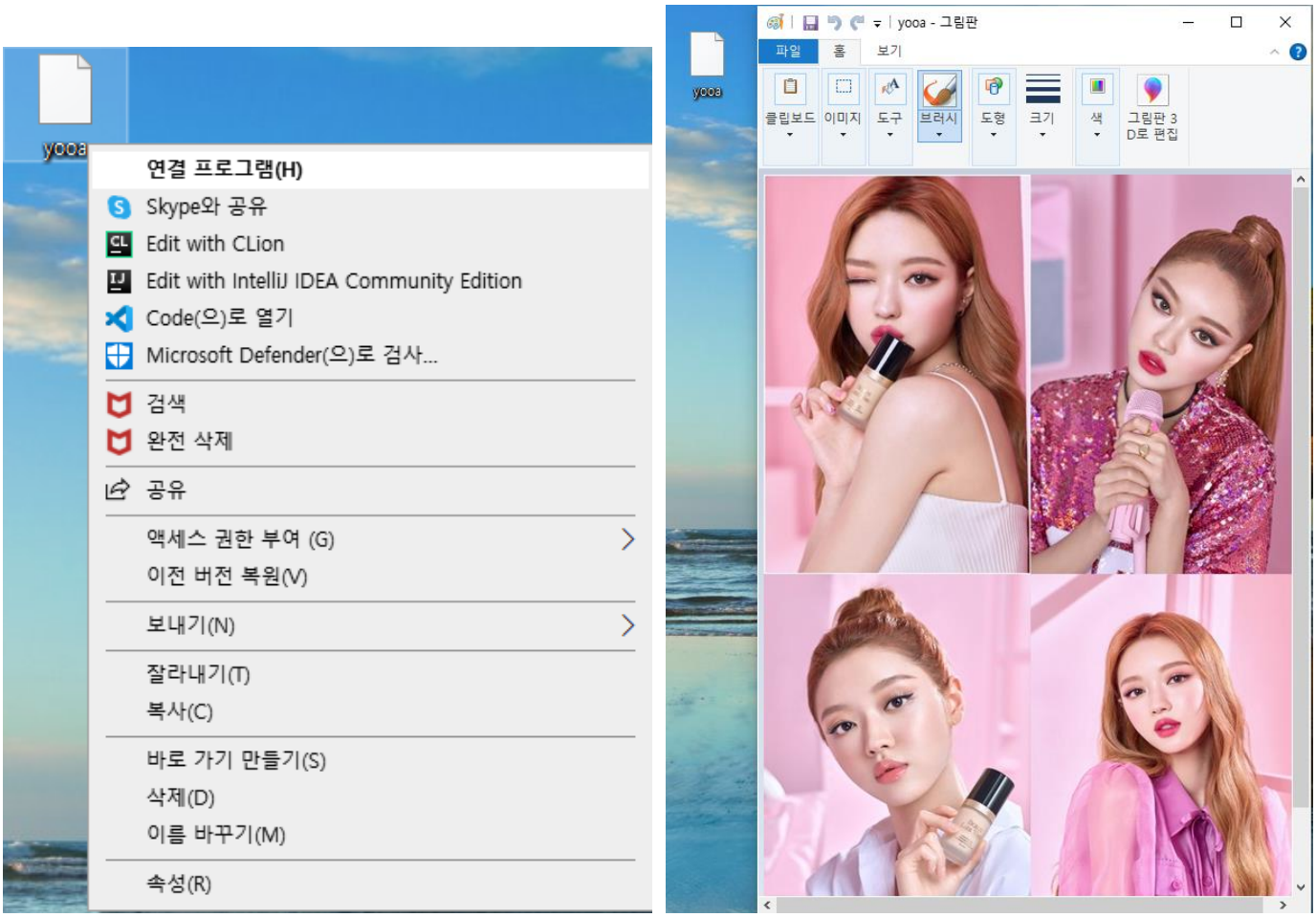


이후, 16 진수 파일을 decode 할 수 있는 Tool 인 HxD 를 이용하여 전체 Response 에서 이미지 데이터만 저장한다. JPG 파일의 Signature 는 16 진수 FF D8 FF E0 이므로 이를 기준으로 윗 부분, 즉 실제 이미지 데이터가 아닌 부분(Status Code, Date 등)을 삭제하여 이미지 데이터만 남겨 저장한다.





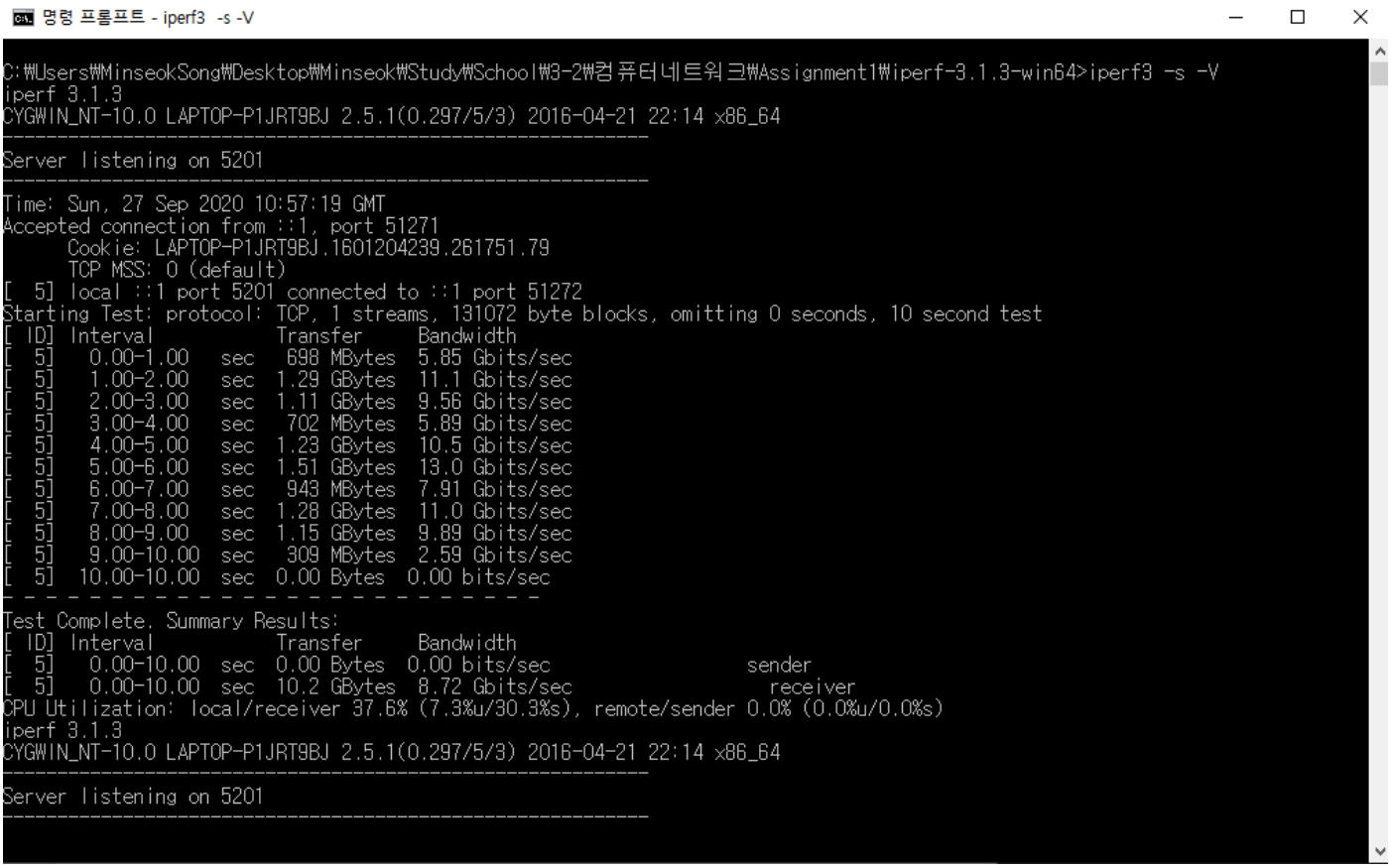
이후 저장된 파일을 그림판, 사진 등의 jpg 호환 프로그램으로 열면 해당 이미지가 저장되었음을 확인할 수 있다.



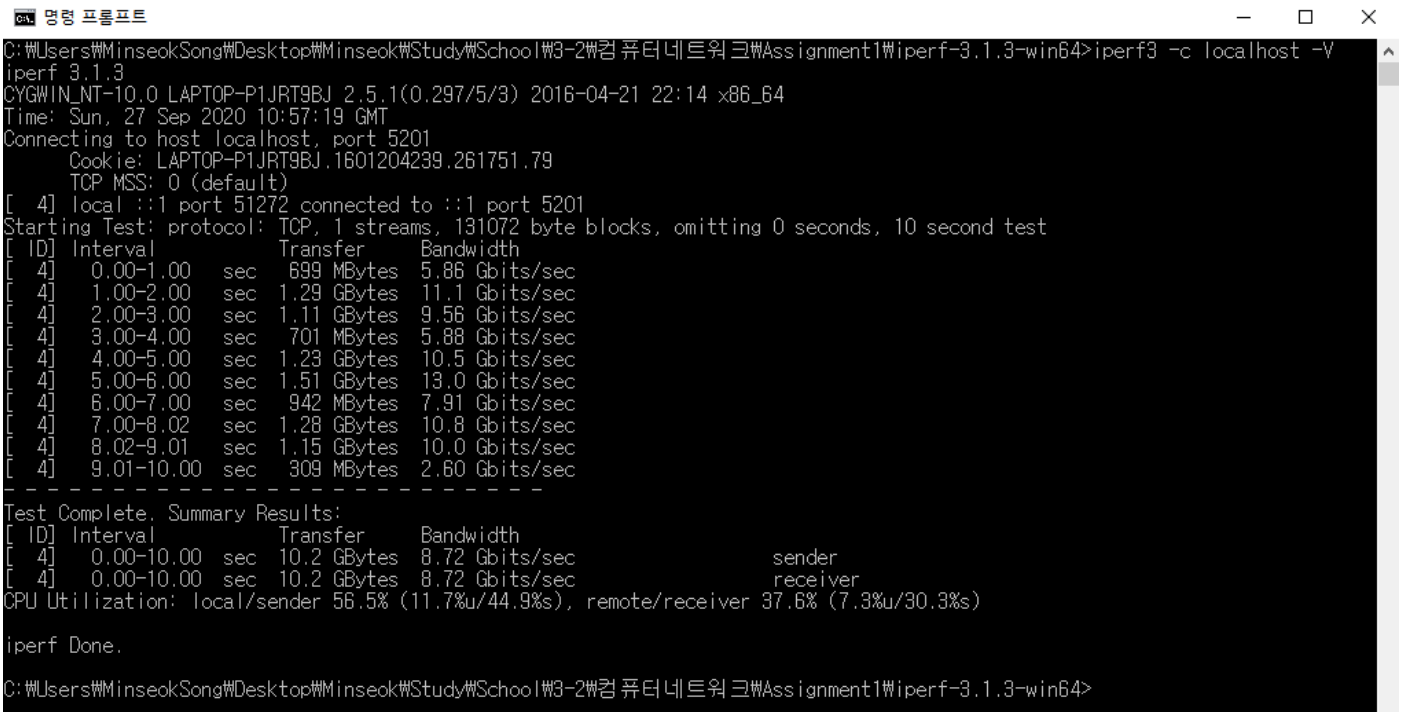
1-2.

Iperf3은 두 컴퓨터(서버 컴퓨터, 클라이언트 컴퓨터)사이의 네트워크 속도를 측정하는 tool이다. 기본으로 TCP 프로토콜을 사용하며, -u옵션을 사용하여 UDP 프로토콜을 이용하여 측정할 수 있다.

Iperf3을 내려 받은 경로에서 실행한다. 먼저 -s옵션을 이용하여 server mode부터 실행한다.



이후 새로운 콘솔에서 -c 옵션을 이용하여 client mode도 실행한다. 연결할 서버의 ip주소 또한 입력한다. 현재 서버 호스트가 localhost이므로 localhost를 입력한다.



두 명령어 모두 -V 옵션을 추가로 이용하여 보다 상세한 정보를 출력하도록 하였다.

현재 server와 client사이 네트워크의 Trasfer, Bandwidth 정보를 10초간 1초 간격 Inverval에 따라 출력한다.

Transfer : 해당 Interval 동안 전송한 데이터 양

Bitrate : 초당 비트 전송률 → Bandwidth : 네트워크의 대역폭(bitrate 단위로 표시)

Cwnd(Congestion Window) : sender 측에서 1 RTT 당 전송되는 packet 의 크기. default 값은 OS 마다 다르다.

또한 -b, -n, -w, -l 4 가지 옵션의 각 기능은 다음과 같다.

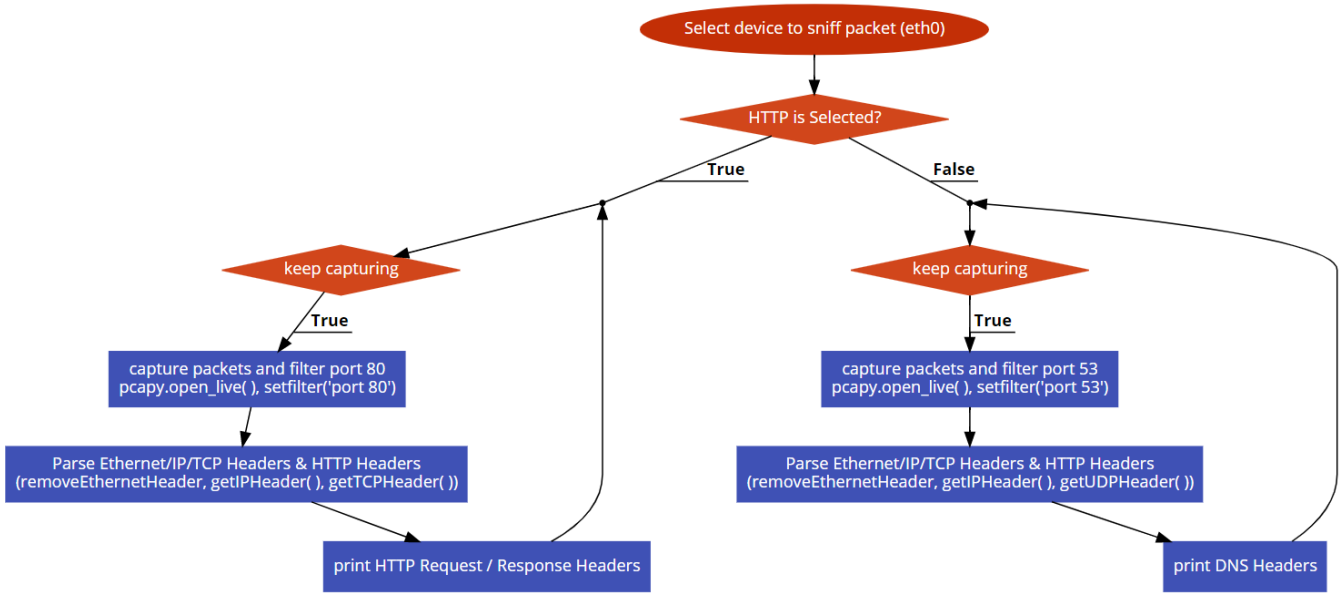
-b (n)	대역폭을 n bits/sec 으로 설정한다. UDP 프로토콜을 사용할 경우 기본값은 1Mbit/sec 이고, TCP 프로토콜을 사용할 경우 제한되어있지 않다. Iperf3 은 기본값으로 TCP 프로토콜을 사용하여 네트워크 속도를 측정한다.
-w (n)	window size / socket buffer size 를 n 으로 설정한다. TCP 프로토콜을 사용할 경우 TCP window size 로 설정된다. 설정된 n 은 server 에서도 적용된다.
-l (n)	Read / Write 할 buffer 의 size 를 n 으로 설정한다. Iperf 는 이 n 개 byte 의 배열을 여러 번 사용함으로써 동작한다. TCP 프로토콜을 사용할 경우 기본값은 128KB 이고, UDP 의 경우 8KB 이다.
-n (m)	Buffer 를 m bytes 만큼 전송하도록 설정한다. 기본적으로 Iperf 는 10 초간 전송하는데 이 m 이 10 초를 넘어가도록 설정되면 10 초를 무시하고 m bytes 를 전송한다.

1-3.

과제에 사용된 언어 및 버전, OS, 라이브러리는 다음과 같다.

Code Language : Python | Version : 3.6.9 |OS : Ubuntu 18.04 | Library : pcapy

project.py 의 Flowchart 는 다음과 같다.



먼저 findalldevs( )를 이용하여 접근 가능한 네트워크 인터페이스 카드 목록을 출력하고, 사용자가 선택하도록 한다. 이후 사용자는 HTTP 패킷을 sniff 할 것인지 DNS 패킷을 sniff 할 것인지 선택하게 된다. 선택된 값에 따라 프로그램이 종료될 때까지 패킷을 캡쳐하여 Ethernet Header, IP Header, TCP/UDP Header 에서 필요한 값을 parsing 한 후 남은 부분을 형식에 맞게 print 한다.

각 단계에 사용된 코드는 다음과 같다.

project.py

```python
# import Libraries(Only pcapy used)
import pcapy

devices = pcapy.findalldevs()

# Ask user to choose the device name to sniff
print ("Available Network Interface Cards are\n ", devices)
dev = int(input("\nSelect Network Device (1 ~ {}) : ".format(len(devices))))-1
dev = devices[dev]
print ("\n", dev, "has Selected")

# Ask user to choose HTTP mode / DNS mode
tp = input("\nSelect HTTP/DNS (1: HTTP / 2: DNS) : ")
print()
```

라이브러리를 import 하고, 사용자에게 device 목록을 보여준 뒤 device 와 HTTP/DNS 를 선택하도록 한다.

project.py

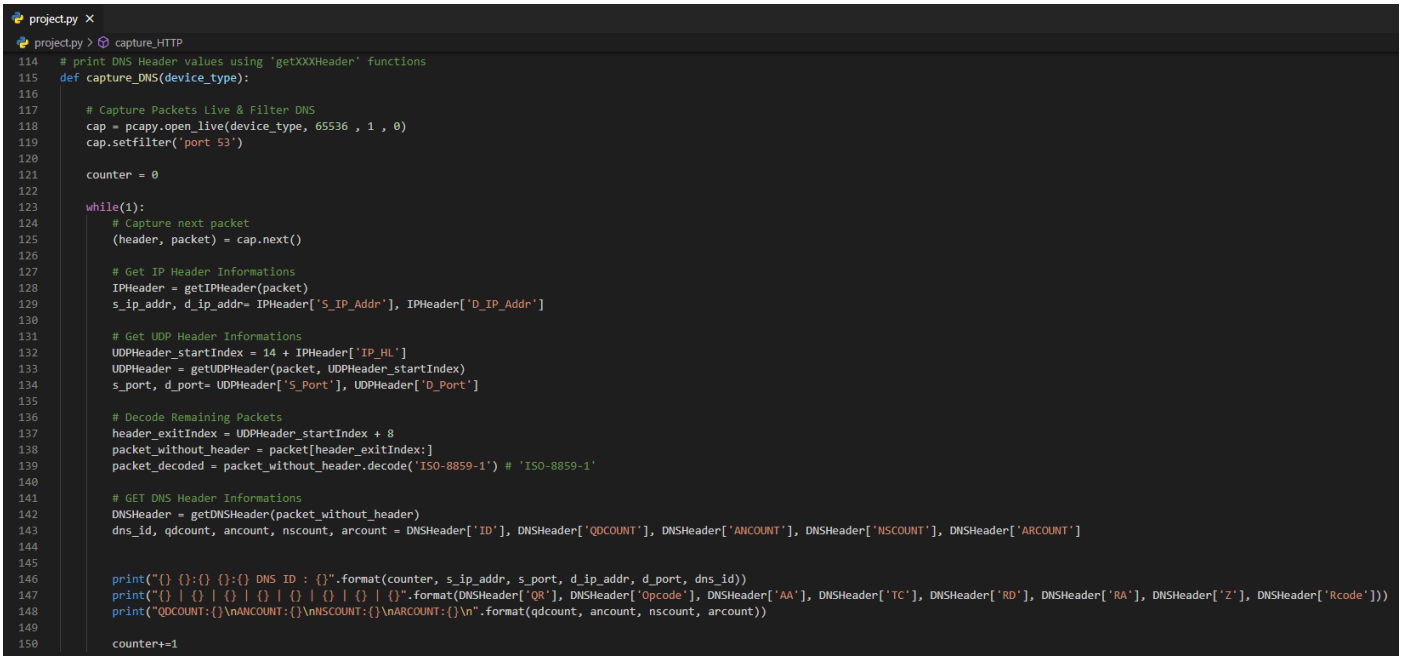
```python
# if User selects 1, call 'capture_HTTP' / if user selects 2, call 'capture_DNS'
if (tp == '1'):
    capture_HTTP(dev)
elif (tp == '2'):
    capture_DNS(dev)
```

사용자가 선택한 값에 따라 HTTP 헤더를 출력하는 함수, DNS 헤더를 출력하는 함수를 실행한다.

project.py

```python
# print HTTP Request/Response Header values using 'getXXXHeader' functions
def capture_HTTP(device_type):

    # Capture Packets Live & Filter HTTP
    cap = pcapy.open_live(device_type, 65536 , 1 , 0)
    cap.setfilter('port 80')

    counter = 0

    while(1):
        # Capture next packet
        (header, packet) = cap.next()

        # Get IP Header Informations
        IPHeader = getIPHeader(packet)
        s_ip_addr, d_ip_addr= IPHeader['S_IP_Addr'], IPHeader['D_IP_Addr']

        # Get TCP Header Informations
        TCPHeader_startIndex = 14 + IPHeader['IP_HL']
        TCPHeader = getTCPHeader(packet, TCPHeader_startIndex)
        s_port, d_port= TCPHeader['S_Port'], TCPHeader['D_Port']

        # Decode Remaining Packets
        header_exitIndex = TCPHeader_startIndex + TCPHeader['Offset']
        packet_without_header = packet[header_exitIndex:]
        packet_decoded = packet_without_header.decode('ISO-8859-1') # 'ISO-8859-1'

        # Check if keyword 'HTTP' contained
        if ('HTTP' in packet_decoded):

            # Check if packet includes Request Header, and print if included
            if ('GET' in packet_decoded) | ('POST' in packet_decoded) | ('PUT' in packet_decoded) | ('DELETE' in packet_decoded) | ('HEAD' in packet_decoded):
                counter+=1
                print("{} {}:{} {}:{} HTTP Request".format(counter, s_ip_addr, s_port, d_ip_addr, d_port))
                index = packet_decoded.index('\r\n\r\n')
                print(packet_decoded[:index] + '\n')

            # Check if packet includes Response Header, and print if included
            else:
                counter+=1
                print("{} {}:{} {}:{} HTTP Response".format(counter, s_ip_addr, s_port, d_ip_addr, d_port))
                index = packet_decoded.index('\r\n\r\n')
                print(packet_decoded[:index] + '\n')
```

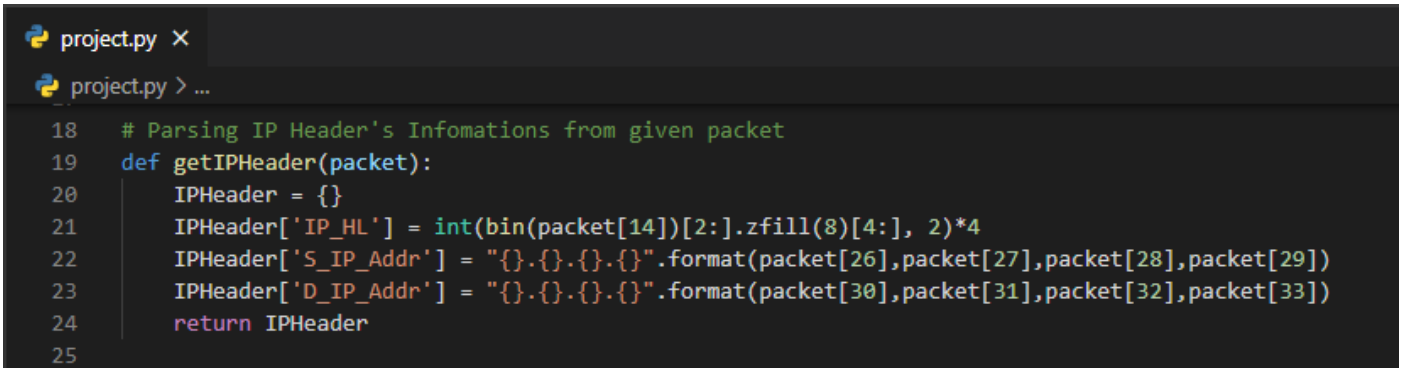
HTTP 헤더를 출력하는 경우 위 함수가 실행된다. 먼저 while 문과 pcapy.open_live( )를 이용해 프로그램이 실행되는 동안 계속 패킷을 캡쳐하도록 하고, setfilter 를 이용하여 80 번 포트, 즉 HTTP 프로토콜만 필터링 하도록 한다. 각 출력마다 번호를 매기기 위해 counter 변수에 선언하고 초기값을 0 으로 할당한다. 이후 next( )를 이용해 패킷에 접근하고, Ethernet Header, IP Header, TCP Header 를 순서대로 parsing 하여 필요한 정보(S_IP, D_IP 등)을 추출한다. 각 parsing 을 위해 별도의 함수를 정의하였으며 이는 뒤에 서술하였다. Ethernet Header 의 경우 IP Header 를 Parsing 하는 과정에서 같이 parse 된다. 각 단계별 Header 를 추출한 후, 패킷의 남은 부분을 decode 하여 HTTP Request / Response 를 구분하고 헤더 부분을 양식에 맞게 출력한다.



```python
# print DNS Header values using 'getXXXHeader' functions
def capture_DNS(device_type):

    # Capture Packets Live & Filter DNS
    cap = pcapy.open_live(device_type, 65536 , 1 , 0)
    cap.setfilter('port 53')

    counter = 0

    while(1):
        # Capture next packet
        (header, packet) = cap.next()

        # Get IP Header Informations
        IPHeader = getIPHeader(packet)
        s_ip_addr, d_ip_addr= IPHeader['S_IP_Addr'], IPHeader['D_IP_Addr']

        # Get UDP Header Informations
        UDPHeader_startIndex = 14 + IPHeader['IP_HL']
        UDPHeader = getUDPHeader(packet, UDPHeader_startIndex)
        s_port, d_port= UDPHeader['S_Port'], UDPHeader['D_Port']

        # Decode Remaining Packets
        header_exitIndex = UDPHeader_startIndex + 8
        packet_without_header = packet[header_exitIndex:]
        packet_decoded = packet_without_header.decode('ISO-8859-1') # 'ISO-8859-1'

        # GET DNS Header Informations
        DNSHeader = getDNSHeader(packet_without_header)
        dns_id, qdcount, ancount, nscount, arcount = DNSHeader['ID'], DNSHeader['QDCOUNT'], DNSHeader['ANCOUNT'], DNSHeader['NSCOUNT'], DNSHeader['ARCOUNT']


        print("{} {}:{} {}:{} DNS ID : {}".format(counter, s_ip_addr, s_port, d_ip_addr, d_port, dns_id))
        print("{} | {} | {} | {} | {} | {} | {} | {}".format(DNSHeader['QR'], DNSHeader['Opcode'], DNSHeader['AA'], DNSHeader['TC'], DNSHeader['RD'], DNSHeader['RA'], DNSHeader['Z'], DNSHeader['Rcode']))
        print("QDCOUNT:{}\nANCOUNT:{}\nNSCOUNT:{}\nARCOUNT:{}\n".format(qdcount, ancount, nscount, arcount))

        counter+=1
```

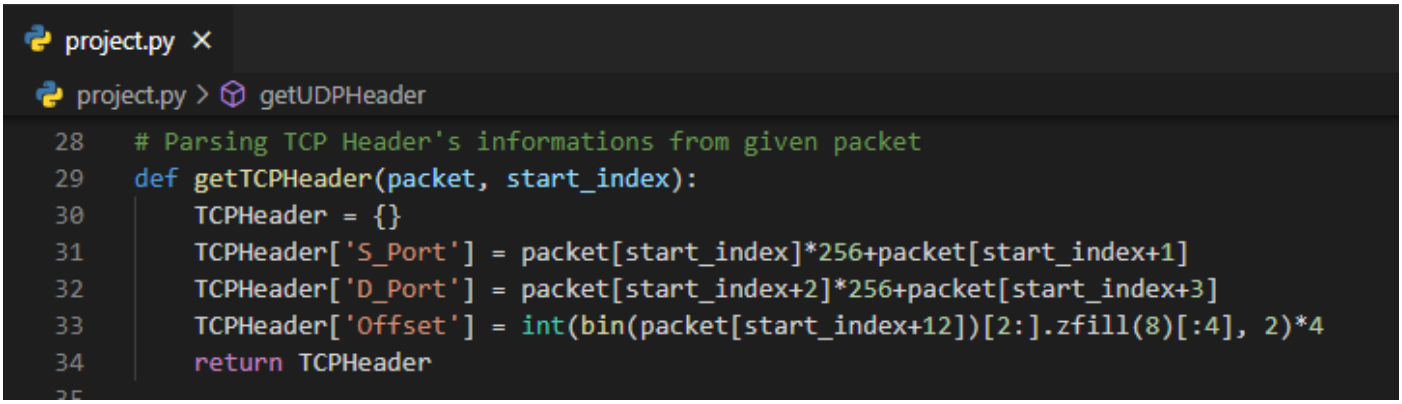
DNS 헤더를 출력하는 과정도 유사하다. 위와 다른 점은 필터링 과정에 port 80 이 아닌 port 53 을 이용하였고, TCP Header 가 아닌 UDP Header 를 parsing 하는 함수를 사용하였다. 이후 DNS Header 정보를 출력하는 과정에 있어 별도의 parsing 과정을 추가로 구현하였다. 마찬가지로 parsing 후에 양식에 맞게 출력한다.

위 과정에서 각 헤더를 parsing 하는데 사용된 'getXXXHeader' 함수들은 다음과 같다.



```python
# Parsing IP Header's Infomations from given packet
def getIPHeader(packet):
    IPHeader = {}
    IPHeader['IP_HL'] = int(bin(packet[14])[2:].zfill(8)[4:], 2)*4
    IPHeader['S_IP_Addr'] = "{}.{}.{}.{}".format(packet[26],packet[27],packet[28],packet[29])
    IPHeader['D_IP_Addr'] = "{}.{}.{}.{}".format(packet[30],packet[31],packet[32],packet[33])
    return IPHeader
```

Ethernet Header 를 건너뛰고 IP Header 를 Parsing 하는 getIPHeader. 이더넷 헤더의 경우 과제에 필요가 없고 그 길이가 고정되어 있어 이와 같이 구현하였다. HTTP / DNS 를 가리지 않고 필요한 함수이다.



```python
# Parsing TCP Header's informations from given packet
def getTCPHeader(packet, start_index):
    TCPHeader = {}
    TCPHeader['S_Port'] = packet[start_index]*256+packet[start_index+1]
    TCPHeader['D_Port'] = packet[start_index+2]*256+packet[start_index+3]
    TCPHeader['Offset'] = int(bin(packet[start_index+12])[2:].zfill(8)[:4], 2)*4
    return TCPHeader
```

TCP Header 를 Parsing 하는 getTCPHeader. 보통 IP Header 의 길이가 20 Byte 이지만 상황에 따라 다르므로, IP Header 의 길이를 나타내는 IP_HL 의 값을 추가로 넘겨받아 패킷을 해당 index 부터 parsing 하도록 하였다. HTTP 의 경우에만 작동한다.

project.py ×

project.py > getUDPHeader

```python
# Parsing UDP Header's informations from given packet
def getUDPHeader(packet, start_index):
    UDPHeader = {}
    UDPHeader['S_Port'] = packet[start_index]*256+packet[start_index+1]
    UDPHeader['D_Port'] = packet[start_index+2]*256+packet[start_index+3]
    return UDPHeader
```

UDP Header 를 parsing 하는 getUDPHeader. 역시 IP Header 의 길이를 넘겨받아 parsing 한다. DNS 의 경우에만 작동한다.

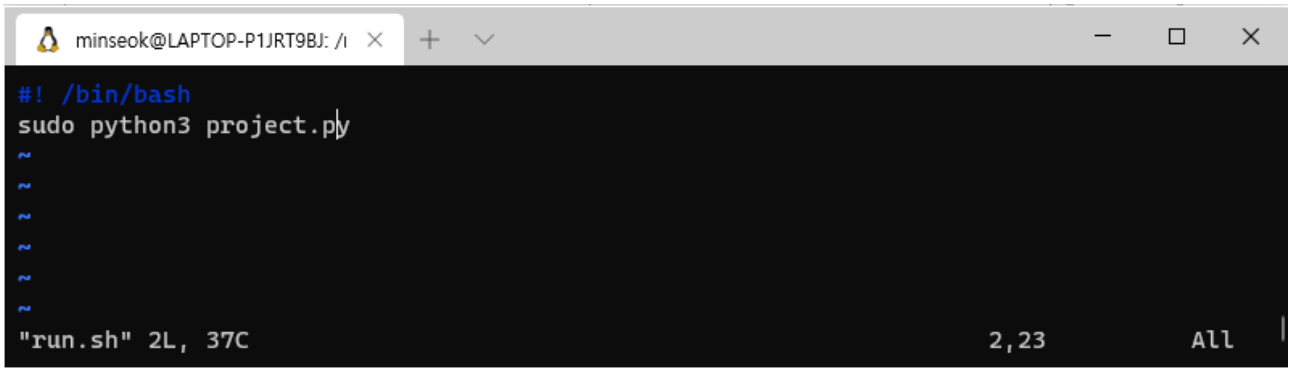
project.py ×

project.py > getUDPHeader

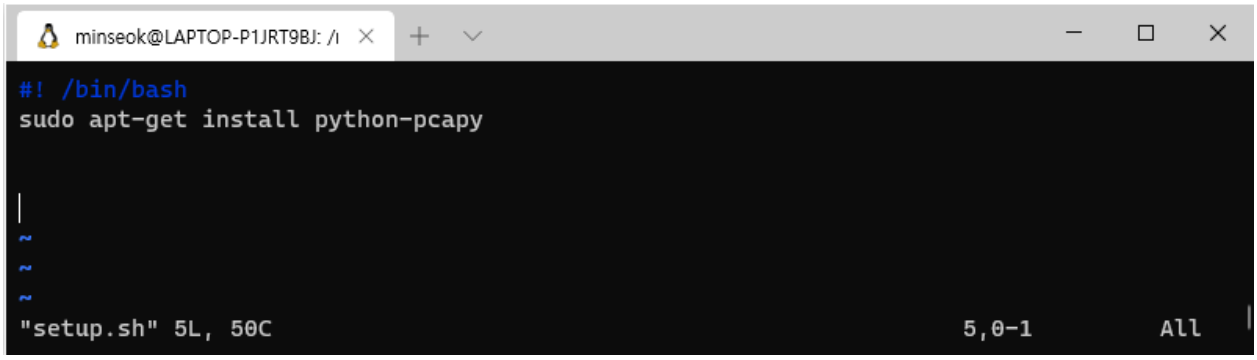
```python

# Parsing DNS Header's informations from given packet
def getDNSHeader(packet_without_header):
    DNSHeader = {}
    DNSHeader['ID'] = hex(packet_without_header[0]*256+packet_without_header[1])[2:]
    DNSHeader['QR'] = bin(packet_without_header[2])[2:].zfill(8)[0]
    DNSHeader['Opcode'] = bin(packet_without_header[2])[2:].zfill(8)[1:5]
    DNSHeader['AA'] = bin(packet_without_header[2])[2:].zfill(8)[5]
    DNSHeader['TC'] = bin(packet_without_header[2])[2:].zfill(8)[6]
    DNSHeader['RD'] = bin(packet_without_header[2])[2:].zfill(8)[7]
    DNSHeader['RA'] = bin(packet_without_header[3])[2:].zfill(8)[0]
    DNSHeader['Z'] = bin(packet_without_header[3])[2:].zfill(8)[1]
    DNSHeader['Rcode'] = bin(packet_without_header[3])[2:].zfill(8)[4:]
    DNSHeader['QDCOUNT'] = packet_without_header[4]*256+packet_without_header[5]
    DNSHeader['ANCOUNT'] = packet_without_header[6]*256+packet_without_header[7]
    DNSHeader['NSCOUNT'] = packet_without_header[8]*256+packet_without_header[9]
    DNSHeader['ARCOUNT'] = packet_without_header[10]*256+packet_without_header[11]
    return DNSHeader
```

DNS Header 를 parsing 하는 getDNSHeader. Pcapy 를 통해 캡쳐된 Packet 은 byte 단위로 구성된 데에 반해 각 정보는 bit 단위로 담겨있어 각각 bin, hex 를 이용하여 변환해주었다.

run.sh 와 setup.sh 의 내용은 다음과 같다.

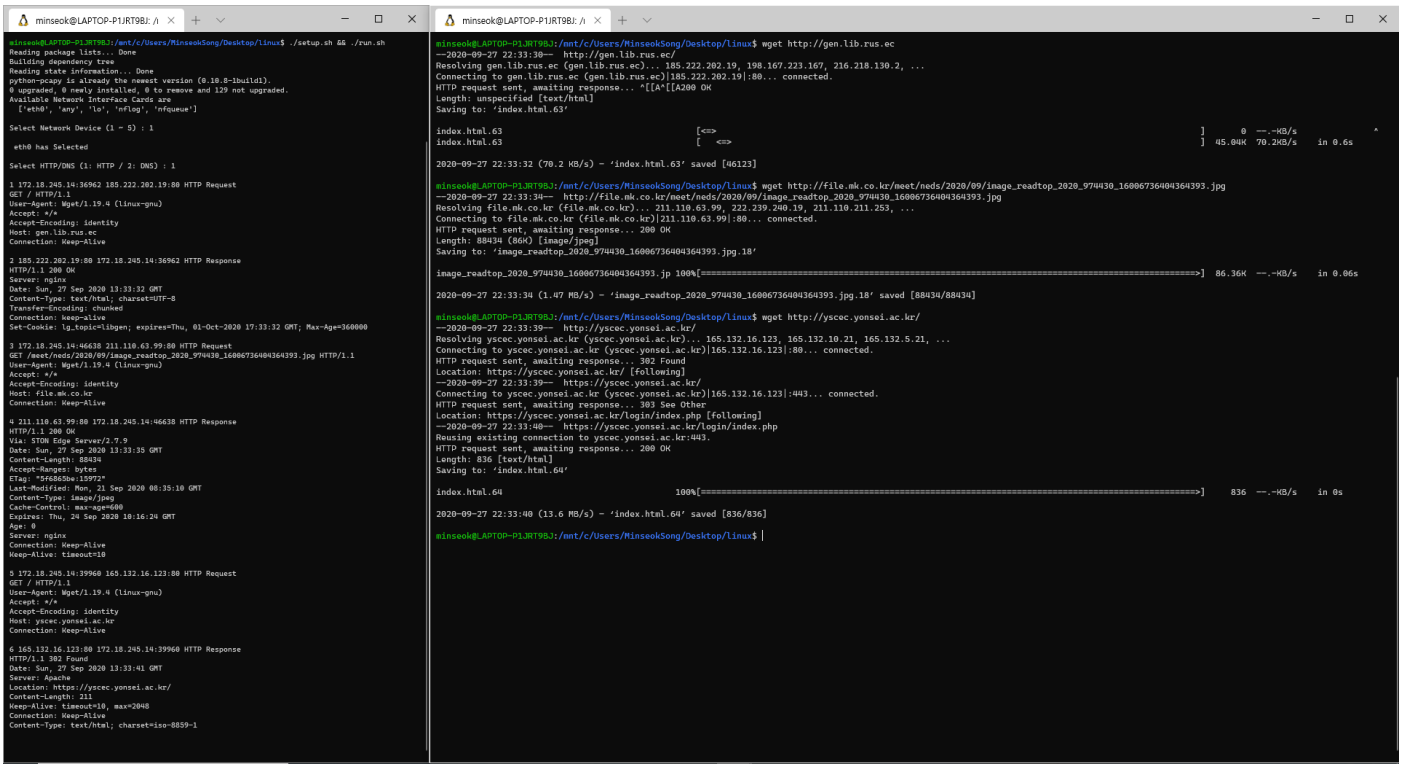


run.sh 는 해당 project.py 파일을 실행한다.

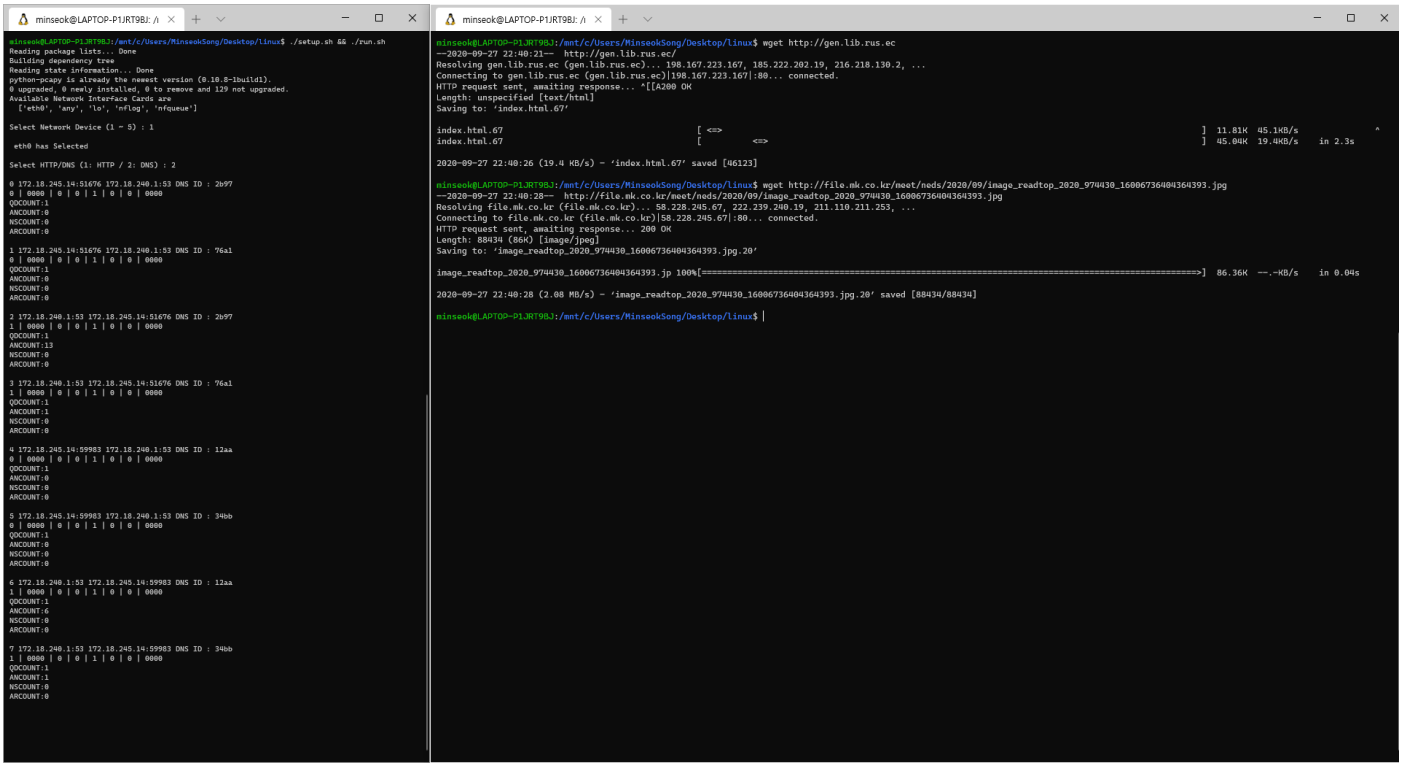


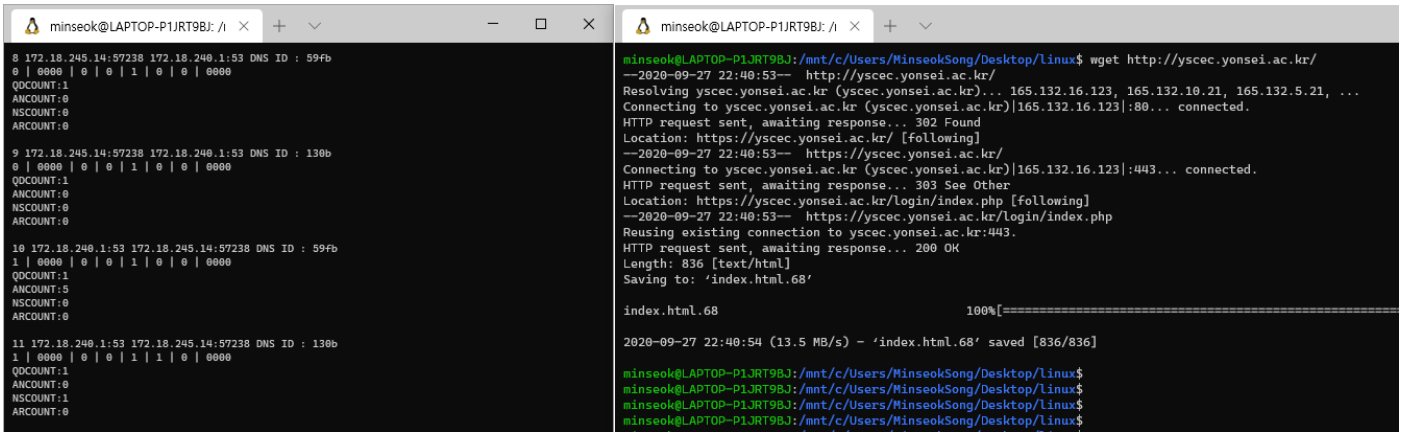
setup.sh 는 실행에 필요한 pcapy 라이브러리를 설치한다.

./setup.sh && ./run.sh 명령을 수행한 후, eth0, HTTP 를 선택한 뒤 3 가지 url 에 wget 명령어로 접근함으로써 수행한 결과는 다음과 같다.



같은 방법으로 실행한 후 DNS 를 선택하여 실행한 과정은 다음과 같다.





http://gen.lib.rus.ec/,
http://file.mk.co.kr/meet/neds/2020/09/image_readtop_2020_974430_16006736404364393.jpg,
http://yscec.yonsei.ac.kr/

사용된 3 가지 URL 은 다음과 같다.