```
#Team 04 UNP Network Monitor (python)
#Christiaan Obbink, Salar Darwish, Ka Yue Sin
#Packet sniffer used to monitor network traffic
#For Linux (tested in Ubuntu)- Sniffs all incoming and outgoing packets
#!/usr/bin/python

import socket, sys
from struct import *
import MySQLdb
import datetime

s = socket.socket()
host = '192.168.56.102'
port = 12345
s.bind((host, port))

s.listen(5)                 # Now wait for client connection.
while True:
    c, addr = s.accept()     # Establish connection with client.
    print 'Got connection from', addr

    try:
        sniffer = socket.socket( socket.AF_PACKET , socket.SOCK_RAW , socket.ntohs(0x0003))
        #Socket = socket gebruikt voor een standaard methode waarmee een programma met een
        ander computerproces communiceert.
        #Socket is een API.
        # raw socket is an internet socket that allows direct sending and receiving of
        Internet Protocol packets without any protocol-specific transport layer formatting.
    except socket.error , msg:
        print 'Socket could not be created. Error Code : ' + str(msg[0]) + ' Message ' + msg[
        1]
        sys.exit()

    #Convert a string of 6 characters of ethernet address into a dash separated hex string
    def eth_addr (a) :
        b = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (ord(a[0]) , ord(a[1]) , ord(a[2]), ord(a[3]),
        ord(a[4]) , ord(a[5]))
        return b

    while True:
        # receive a packet
        packet = sniffer.recvfrom(65565)
        #packet string from tuple
        packet = packet[0]

        #----------------------------parse ethernet
        header---------------------------------------------------------------
        eth_length = 14

        eth_header = packet[:eth_length]
        eth = unpack('!6s6sH' , eth_header)
        eth_protocol = socket.ntohs(eth[2])

        #-------------------------Database connection--------------------------
        now = datetime.datetime.now()
        logtime = now.strftime("%d-%m-%Y %H:%M")
        try:
```

```
    # Open database connection
        con = MySQLdb.connect("localhost","root","Welkom01","UNP" )

    # prepare a cursor object using cursor() method
        sql = con.cursor()
    # Prepare SQL query to INSERT a record into the database.
        sql.execute("""INSERT INTO ETH(Datetime,Dest_mac,Source_mac,Protocol)
    VALUES (%s, %s, %s, %s)""",(logtime,eth_addr(packet[0:6]),eth_addr(packet[6:12]),
    eth_protocol))
        con.commit()

        print 'Succesfully Inserted the ETHERNET values to DB !\n'
    except MySQLdb.Error, e:
        print "Error %d: %s" % (e.args[0],e.args[1])
        sys.exit(1)
    finally:
        if con:
            con.close()
    #-------------------------Database connection-------------------------

    print 'ETHERNET Packet \n' + 'Destination MAC:' + eth_addr(packet[0:6]) + ' | Source
    MAC:' + eth_addr(packet[6:12]) + ' | Protocol:' + str(eth_protocol)

    #------------------------------Parse IP packets, IP Protocol number =
    8-------------------------------------
    if eth_protocol == 8 :
        #Parse IP header
        #take first 20 characters for the ip header
        ip_header = packet[eth_length:20+eth_length]

        #now unpack them :)
        iph = unpack('!BBHHHBBH4s4s' , ip_header)

        version_ihl = iph[0]

        #To the the ip version we have to shift
        #the first element 4 bits right. Because in the first element
        #is stored the ip version and the header lenght in this way
        #first four bits are ip version and the last 4 bites are
        #the header lenght
        version = version_ihl >> 4  #The first header field in an IP packet is the
        four-bit version field. For IPv4,
        #this has a value of 4 (hence the name IPv4).

        #ihl is the IP HEADER LENTGH
        #Now to get the header lenght we use "and" operation to make the
        #Ip versional bits equal to zero, in order to the the desired data
        ihl = version_ihl & 0xF #The second field (4 bits) is the Internet Header Length
        (IHL),
        #which is the number of 32-bit words in the header. Since an IPv4 header may
        contain a variable number of options,
        #this field specifies the size of the header (this also coincides with the
        offset to the data)

        iph_length = ihl * 4

        ttl = iph[5] #waarde de tijd dat een hoeveelheid data in het netwerk kan bestaan
```

```
voordat het wordt weggegooid
protocol = iph[6] #This field defines the protocol used in the data portion of
the IP datagram.
s_addr = socket.inet_ntoa(iph[8]); #the sender of the packet.
d_addr = socket.inet_ntoa(iph[9]); #the sender of the packet.

#checksum The 16-bit checksum field is used for error-checking of the header.
When a packet arrives at a router,
#the router calculates the checksum of the header and compares it to the
checksum field.
#If the values do not match, the router discards the packet.

#Total length : This 16-bit field defines the entire packet (fragment) size,
including header and data, in bytes.

#identification: This field is an identification field and is primarily used for
uniquely identifying the group of fragments of a single IP datagram

#he fragment offset field, measured in units of eight-byte blocks (64 bits), is
13 bits long and specifies the offset of a particular fragment relative to
#the beginning of the original unfragmented IP datagram.
#------------------------Database connection-------------------------
now = datetime.datetime.now()
logtime = now.strftime("%d-%m-%Y %H:%M")
try:
# Open database connection
    con = MySQLdb.connect("localhost","root","Welkom01","UNP" )

# prepare a cursor object using cursor() method
    sql = con.cursor()
# Prepare SQL query to INSERT a record into the database.
    sql.execute("""INSERT INTO
    IP(Datetime,version,IHL,TTL,Protocol,Source_addr,Dest_addr)
VALUES (%s, %s, %s, %s, %s, %s, %s)""",(logtime,version,ihl,ttl,protocol,s_addr,
d_addr))
    con.commit()

    print 'Succesfully Inserted the IP values to DB !\n'
except MySQLdb.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
    sys.exit(1)
finally:
    if con:
        con.close()
#------------------------Database connection-------------------------

print 'IP Packet \n' + 'Version:' + str(version) + ' | IP Header Length:' + str(
ihl) + ' | TTL:' + str(ttl) + ' | Protocol:' + str(protocol) + ' | Source
Address:' + str(s_addr) + ' | Destination Address:' + str(d_addr)

#-------------------------------------TCP
protocol-----------------------------------------
if protocol == 6 :
    t = iph_length + eth_length
    tcp_header = packet[t:t+20]

    #now unpack them :)
```

```
        tcph = unpack('!HHLLBBHHH' , tcp_header)

        source_port = tcph[0] #port van waar verstuurd wordt
        dest_port = tcph[1] #port van waar naar verstuur word
        sequence = tcph[2] #Een getal dat door een partij bij het maken van de
        verbinding vrijwel willekeurig gegenereerd wordt,
        #waarna het door die partij de rest van de sessie gebruikt wordt om aan te
        geven dat het om diezelfde sessie gaat.
        acknowledgement = tcph[3] # Een getal dat aangeeft welk segment van het
        laatste ontvangen pakket ontvangen is.
        doff_reserved = tcph[4] #Dit veld is gereserveerd voor eventuele
        uitbreidingen in de toekomst.
        tcph_length = doff_reserved >> 4 #De lengte van de headers om verlies te
        controleren en om aan te geven waar de data precies begint.
        #De headerlengte heeft een minimum lengte van 5 en maximum lengte van 15.
        Dit bepaalt het aantal 32-bit woorden.
        #(Een woord = een rij). Aangezien de opties 0 kan zijn, is het minimum 5 en
        het maximum kan 15 zijn.

        #checksum =  Een getal dat afhangt van het hele pakket, om de inhoud van het
        hele pakket te kunnen controleren.
        #windowsize = De grootte van het leesvenster dat over de verbinding
        "schuift", dit dient voor de Flow control.
        #De ontvanger kan aangeven hoeveel bytes hij wil ontvangen. Dit dient om
        overbelasting te voorkomen.
        #Options = (variabel aantal bytes, daarom is bekendmaking van de
        headerlengte nodig): Allerlei aanvullende opties, zoals timestamping.
        #Data = de daadwerkelijke gegevens (ingecapsuleerde applicatie protocol data).

        #poortnummers:7 echo (stuurt terug wat ontvangen werd),13 daytime (geeft
        huidige datum en tijd),20 FTP (dataconnectie)
        #21 FTP (commandoconnectie),22,SSH,23,Telnet,25 SMTP,53 DNS,80 World Wide
        Web HTTP,88 Kerberos,110 POP3
        #137 Netbios Name Service,143 IMAP,389 LDAP,443 HTTPS,5222 XMPP


    #-------------------------Database connection--------------------------
        now = datetime.datetime.now()
        logtime = now.strftime("%d-%m-%Y %H:%M")
        try:
        # Open database connection
            con = MySQLdb.connect("localhost","root","Welkom01","UNP" )

        # prepare a cursor object using cursor() method
            sql = con.cursor()
        # Prepare SQL query to INSERT a record into the database.
            sql.execute("""INSERT INTO
            TCP(Datetime,Source_port,Desc_port,Sequence,Acknowledge,Length)
        VALUES (%s, %s, %s, %s, %s, %s)""",(logtime,source_port,dest_port,sequence,
        acknowledgement,tcph_length))
            con.commit()

            print 'Succesfully Inserted the TCP values to DB !\n'
        except MySQLdb.Error, e:
            print "Error %d: %s" % (e.args[0],e.args[1])
            sys.exit(1)
        finally:
```

```
            if con:
                con.close()
        #------------------------Database connection-------------------------

            print 'TCP Packet: \n' + 'Source Port:' + str(source_port) + ' | Dest Port:'
            + str(dest_port) + ' | Sequence Number:' + str(sequence) + ' |
            Acknowledgement:' + str(acknowledgement) + ' | TCP header length:' + str(
            tcph_length)

            h_size = eth_length + iph_length + tcph_length * 4
            data_size = len(packet) - h_size

            #get data from the packet
            data = packet[h_size:]

            print 'Data : ' + data

        #---------------------------------ICMP
        Packets----------------------------------------
        elif protocol == 1 :
            u = iph_length + eth_length
            icmph_length = 4
            icmp_header = packet[u:u+4]

            #now unpack them :)
            icmph = unpack('!BBH' , icmp_header)

            icmp_type = icmph[0] # Het ICMP Type bepaalt het soort bericht
            code = icmph[1] #s een manier om een type verder op te delen, indien nodig.
            #Zo wordt destination unreachable verder opgedeeld in port unreachable, host
            unreachable e.a.,
            #terwijl er aan de andere kant geen verdere opdeling is van echo request.
            checksum = icmph[2] #De checksum wordt berekend over het hele te verzenden
            ICMP-pakket

            #------------------------Database connection-------------------------
            now = datetime.datetime.now()
            logtime = now.strftime("%d-%m-%Y %H:%M")
            try:
            # Open database connection
                con = MySQLdb.connect("localhost","root","Welkom01","UNP" )

            # prepare a cursor object using cursor() method
                sql = con.cursor()
            # Prepare SQL query to INSERT a record into the database.
                sql.execute("""INSERT INTO ICMP(Datetime,Type,Code,Checksum)
            VALUES (%s, %s, %s, %s)""",(logtime,icmp_type,code,checksum))
                con.commit()

                print 'Succesfully Inserted the ICMP values to DB !\n'
            except MySQLdb.Error, e:
                print "Error %d: %s" % (e.args[0],e.args[1])
                sys.exit(1)
            finally:
                if con:
                    con.close()
            #------------------------Database connection-------------------------
```

```
        print 'ICMP Packet: \n' + 'Type:' + str(icmp_type) + ' | Code:' + str(code) +
         ' | Checksum:' + str(checksum)

        h_size = eth_length + iph_length + icmph_length
        data_size = len(packet) - h_size

        #get data from the packet
        data = packet[h_size:]

        print 'Data : ' + data

    #---------------------------UDP packets-------------------------------------
    elif protocol == 17 :
        u = iph_length + eth_length
        udph_length = 8
        udp_header = packet[u:u+8]

        #now unpack them :)
        udph = unpack('!HHHH' , udp_header)

        source_port = udph[0] #port van waar verstuurd wordt
        dest_port = udph[1] #port van waar naar verstuur word
        length = udph[2]  #The length of the entire UDP datagram, including both
        header and Data fields.
        checksum = udph[3] #een controle voor de integriteit van de data en het is
        16 bits
        #pesudo header = de source port en dest port en protocol en length samen

    #-------------------------Database connection--------------------------
        now = datetime.datetime.now()
        logtime = now.strftime("%d-%m-%Y %H:%M")
        try:
        # Open database connection
            con = MySQLdb.connect("localhost","root","Welkom01","UNP" )

        # prepare a cursor object using cursor() method
            sql = con.cursor()
        # Prepare SQL query to INSERT a record into the database.
            sql.execute("""INSERT INTO
            UDP(Datetime,Source_port,Dest_port,Length,Checksum)
        VALUES (%s, %s, %s, %s, %s)""",(logtime,source_port,dest_port,length,checksum
        ))
            con.commit()

            print 'Succesfully Inserted the UDP values to DB !\n'
        except MySQLdb.Error, e:
            print "Error %d: %s" % (e.args[0],e.args[1])
            sys.exit(1)
        finally:
            if con:
                con.close()
    #-------------------------Database connection--------------------------

        print 'UDP Packet: \n' + 'Source Port:' + str(source_port) + ' | Dest Port:'
        + str(dest_port) + ' | Length:' + str(length) + ' | Checksum:' + str(checksum)
```

```
            h_size = eth_length + iph_length + udph_length
            data_size = len(packet) - h_size

            #get data from the packet
            data = packet[h_size:]

            print 'Data : ' + data

        #-------------------------some other IP packet like
        IGMP-----------------------------
        else :
            print 'Protocol other than TCP/UDP/ICMP'
            #ip v6
            #version: Same as ipv4 header
            #Hop Limit : Replaces the time to live field of IPv4.
            #Source Address (128 bits) The IPv6 address of the sending node.
            #Destination Address (128 bits) The IPv6 address of the destination node(s).
            #Next Header:The 8-bit Next Header field identifies the type of header
            immediately following the IPv6 header and
            #located at the beginning of the data field (payload) of the IPv6 packet.
            #flow lable:  in the IPv6 header. A source can use this field to label those
            packets for which the source requests special handling by the IPv6 routers.
            #For example, a source can request non-default quality of service or
            real-time service
            #payload length:The 16-bit payload length field contains the length of the
            data field in octets/bits following the IPv6 packet header.
            #The 16-bit Payload length field puts an upper limit on the maximum packet
            payload to 64 kilobytes.
            #Traffic class:The 8-bit Priority field in the IPv6 header can assume
            different values to enable the source
            #node to differentiate between the packets generated by it by associating
            different delivery priorities to them.

    try:
        c.send("connected")
    except:
        break
c.close
```