Laborator 11
Dezvoltare aplicații pentru proxy snif&decode

Proxy

Termenul de intermediar (proxy) l-am mai discutat pana acum la diverse materii. Însă până acum nu l-am avut sub control direct (deci să fie intermediarul nostru). De multe ori în cazul în care știi că te afli sub un posibil interes de la instrumente automate sau atacuri manuale este o idee bună să oferi și câteva momeli. În acest context crearea unui proxy aflat sub control direct care permite orice suită suplimentară de acțiuni în mod automat este necesară. În consecință în acest laborator vom discuta un exemplu simplu de creare a unei astfel de aplicații. Ca observație - aș fi ales alt limbaj în loc de Python dar având în vedere publicul țintă merge și acesta.

Programul este destul de simplu. În primul rand se dorește afișarea comunicării dintre mașina locală și cea aflată la distanță. După cum am învățat la paradigme aplicarea modelului proxy presupune introducerea unui intermediar într-un flux uni- sau bidirecțional de date. Ca atare se pornește de la o abordare simplă de aplicație client server bazată pe TCP IP peste care introducem posibilitatea modificării fluxului de date dar în același timp acest lucru ne permite să desfășurăm oricâte alte operații (specifice echipei albastre sau roșii) asupra fluxului dar și a mașinii(lor) aflate la distanță.

Pentru analiza mai detailată a comunicațiilor ar trebui creat ceva în genul unui hexdump ca în exemplul de mai jos (asta pentru început).

```
import sys
import socket
import threading
HEX_FILTER = ''.join([(len(repr(chr(i))) == 3) and chr(i) or '.' for i in range(256)])
def hexdump(src, length=16, show=True):
    if isinstance(src, bytes):
src = src.decode()
results = list()
for i in range(0, len(src), length):
    word = str(src[i:i+length])
    printable = word.translate(HEX_FILTER)
hexa = ' '.join([f'{ord(c):02X}' for c in word])
hexwidth = length*3
    results.append(f'{i:04x} {hexa:<{hexwidth}} {printable}')
if show:
for line in results:
print(line)
else:
return results
```

Aici se creează un șir HEXFILTER care conține caracterele afișabile din ASCII sau dacă nu (de control de ex). Modificați (:)) și testați programul.

Această funcție ne va permite monitorizarea vizuală (mod text) a comunicației care trece prin proxy.

```
def receive_from(connection):
buffer = b""
  connection.settimeout(5)
try:
```

```
while True:
   data = connection.recv(4096)
if not data:
break
buffer += data
except Exception as e:
pass
return buffer
```

Pentru a recepționa datele local sau la distanță se trimite un obiect la sochet. Se observă că s-a utilizat un tampon. Celele 5 secunde sunt pentru situația în care conexiunile sunt de mare viteză altfel ar trebui să mai creșteți această durată în mod corespunzător. La sfârșit retrimitem datele către apelant. Uneori se dorește modificarea pachetelor de cerere sau de răspuns la nivelul proxy și pentru aceasta s-au adăugat două funcții dedicate *request_handler* și *response_handler*.

```
def request_handler(buffer):
# modificari asupra pachetelor
return buffer
def response_handler(buffer):
# modificari asupra pachetelor
return buffer
```

Aici se pot modifica pachete, realiza chiar analiza acestora sau orice altceva este necesar.

sudo python proxy.py adresa_ip 21 adresa_ftp 21 True

Programul fără operații în proxy

```
import sys
import socket
import threading

def hexdump(src, length=16):
    result = []
    digits = 4 if isinstance(src, str) else 2
    for i in range(0, len(src), length):
        s = src[i:i + length]
        hexa = b' '.join([b"%0*X" % (digits, ord(x)) for x in s])
        text = b''.join([x if 0x20 <= ord(x) < 0x7F else b'.' for x in s])
        result.append( b"%04X   %-*s   %s" % (i, length * (digits + 1), hexa, text))

    print(b'\n'.join(result))
def receive_from(connection):
    buffer = b""
    # intarziere de doua secunde - se poate modifica dupa necesitati :)
    connection.settimeout(2)
    try:
        while True:
            data = connection.recv(4096)
            if not data:
                break
            buffer += data
    except TimeoutError:
        pass
```

```
        return buffer
# schimbarea oricarei cereri destinate unei alte masini
def request_handler(buffer):
        # se realizeaza modificarea pachetului.....
        return buffer
# se modifica orice raspuns de    la masina gazda/locala
def response_handler(buffer):
        # iar modific pachetul ....
        return buffer
def proxy_handler(client_socket, remote_host, remote_port, receive_first):
        # conectarea la alta masina
        remote_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        remote_socket.connect((remote_host, remote_port))
        # primire date si control flux
        if receive_first:
                remote_buffer = receive_from(remote_socket)
                hexdump(remote_buffer)
                # trimitere catre gestionarul propriu a raspunsului
                remote_buffer = response_handler(remote_buffer)

                # daca sunt date care trebuie trimise clientului local atunci se vor trimite
                if len(remote_buffer):
                        print("[<==] Trimit %d octeti catre clientul local." % len(remote_buffer))
                        client_socket.send(remote_buffer)
        # repetam procesul
        while True:
                # citim de la clientul local
                local_buffer = receive_from(client_socket)
                if len(local_buffer):
                        print("[==>] Am primit %d octeti de la clientul local." % len(local_buffer))
                        hexdump(local_buffer)
                        # se trimit octetii catre gestionarul local
                        local_buffer = request_handler(local_buffer)
                        # se trimit datele catre masina externa
                        remote_socket.send(local_buffer)
                        print("[==>] Trimit catre exterior")
                # primire raspuns
                remote_buffer = receive_from(remote_socket)
                if len(remote_buffer):
                        print("[<==] Am primit %d octeti din afara" % len(remote_buffer))
                        hexdump(remote_buffer)
                        # ii trimit catre gestionarul nostru local
                        remote_buffer = response_handler(remote_buffer)
                        # trimit raspunsul catre soclul local
                        client_socket.send(remote_buffer)
                        print("[<==] Trimisi local.")
                # daca nu mai am comunicare cu exteriorul inchid conexiunea
                if not len(local_buffer) or not len(remote_buffer):
                        client_socket.close()
                        remote_socket.close()
                        print("[*] Nu mai sunt date inchid conexiunea.")
                        break
def server_loop(local_host, local_port, remote_host, remote_port,
                        receive_first):
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    try:
        server.bind((local_host, local_port))
    except socket.error as exc:
        print("[!!] Nu pot asculta %s:%d" % (local_host,
                                             local_port))
        print("[!!] Incercati alte porturi sau modificati" "permisiunile sistemului.")
        print(f"[!!] A aparut urmatoarea eroare: {exc}")
        sys.exit(0)
    print("[*] Ascult pe %s:%d" % (local_host, local_port))
    server.listen(5)
    while True:
        client_socket, addr = server.accept()

        print("[==>] Primesc date de la %s:%d" % ( addr[0], addr[1]))

        # pornesc un fir pentru gestiunea conexiunii cu masina din afara
        proxy_thread = threading.Thread(target=proxy_handler, args=(
            client_socket, remote_host, remote_port, receive_first))
        proxy_thread.start()

if __name__ == '__main__':
    if len(sys.argv[1:]) != 5:
        print("Utilizare: ./proxy.py [localhost] [localport] [remotehost] "
              "[remoteport] [receive_first]")
        print("De exemplu: ./proxy.py 127.0.0.1 9000 10.12.132.1 9000 True")
        sys.exit(0)
    # stabilesc parametrii clientului local
    local_host = sys.argv[1]
    local_port = int(sys.argv[2])

    # datele masinii din afara
    remote_host = sys.argv[3]
    remote_port = int(sys.argv[4])

    # devierea fluxului prin intermediarul nostru inainte de a-l retrimite afara
    receive_first = sys.argv[5]

    if "True" in receive_first:
        receive_first = True
    else:
        receive_first = False
    # activare soclu ascultare
    server_loop(local_host, local_port, remote_host, remote_port, receive_first)
```

Tema

Utilizați scapy pentru a completa cele două zone de modificare a fluxului prevazute în program

Un ex de utilizare a scapy

```
hostinput = str(input("Dati un IP sau un FQDN: "))
# FQDN - Fully Qualified Domain Name

#crearea functiei proprii si intoarcerea valorii calcfoo
def chowrecon(arg1):
    os.system("nslookup" + ' ' + arg1)
```

```
    os.system("ping" + ' ' + arg1)
    calcfoo = ("trimit ceva la stdout: " + arg1)
    return calcfoo
foo = chowrecon(hostinput)
print(foo)
```

Dacă se lucrează cu mai multe straturi din iso-osi atunci pui modul scapy în proiect și

```
from scapy.all import *

if __name__ == '__main__':
    ip_layer = scapy.IP(dst="192.168.0.1") #primul nivel
    icmp_layer = scapy. ICMP(seq=9999) #al doilea nivel
    packet = ip_layer / icmp_layer #combinare niveluri
    send(packet) #emitere pachet
```

PING

Întrucât la un alt laborator v-am pus să citiți din documentație pentru a vă readuce aminte IPv4/IPv6 pentru a putea înțelege mai bine și fixa aceste cunoștințe aveți mai jos un exemplu care simulează un ping utilizând funcții clasice de conectare bazate pe socluri. NU uitați vă rog că trebuie drepturi de admin deci din terminalul proiectului pycharm veți da

sudo python3 nume_fisier.py

pentru testare

```
import time
import socket
import struct
import select
import random
import asyncore

ICMP_ECHO_REQUEST = 8

ICMP_CODE = socket.getprotobyname('icmp')
ERROR_DESCR = {
    1: ' - icmp numai ca root'
        'lansati din terminalul proiectului programul cu sudo pytheon3 nume sursa',
    10013: ' - icmp poate fi transmis'
            ' numai de utilizatori sau procese cu drept de root'
}
__all__ = ['create_packet', 'do_one_ping', 'verbose_ping', 'PingQuery',
           'multi_ping_query']
def checksum(source_string):
    sum = 0
    count_to = (len(source_string) / 2) * 2
    count = 0
    while count < count_to:
        this_val = source_string[count + 1] * 256 + source_string[count]
        sum = sum + this_val
        sum = sum & 0xffffffff
```

```
            count = count + 2
        if count_to < len(source_string):
            sum = sum + ord(source_string[len(source_string) - 1])
            sum = sum & 0xffffffff
        sum = (sum >> 16) + (sum & 0xffff)
        sum = sum + (sum >> 16)
        answer = ~sum
        answer = answer & 0xffff
        answer = answer >> 8 | (answer << 8 & 0xff00)
        return answer
def create_packet(id):
        #se creaza un nou pachet de tip ecou in functie de un id
        # Header-ul este de tipul (8), codul (8), checksum (16), id (16), secventa (16)
        header = struct.pack('bbHHh', ICMP_ECHO_REQUEST, 0, 0, id, 1)
        data = 192 * 'Q'
        # se calculeaza suma de control pentru headerul fals/dummy
        my_checksum = checksum(header + data.encode('ASCII'))
        # avand suma de control corecta este mai usor sa cream un nou header/cap
pe care sa il adaugam ulterior
        header = struct.pack('bbHHh', ICMP_ECHO_REQUEST, 0,
                                        socket.htons(my_checksum), id, 1)
        return header + data.encode('ASCII')
def do_one_ping(dest_addr, timeout=1):
        #trimite un singur ping catre o singura destinatie (ip/nume gazda)
        try:
            my_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW,
ICMP_CODE)
        except socket.error as e:
            if e.errno in ERROR_DESCR:
                # Operation not permitted
                raise socket.error(''.join((e.args[1], ERROR_DESCR[e.errno])))
            raise    # raise the original error
        try:
            host = socket.gethostbyname(dest_addr)
        except socket.gaierror:
            return
        # mentin valoarea intre 0 si 65535 pentru id (limitari os)
        packet_id = int((id(timeout) * random.random()) % 65535)
        packet = create_packet(packet_id)
        while packet:
            # pentru ca de fapt icmp nu utilizeaza un port dar functiile
            # utilizate in simulare asteapta asa ceva le vom furniza un port fals
            sent = my_socket.sendto(packet, (dest_addr, 1))
            packet = packet[sent:]
        delay = receive_ping(my_socket, packet_id, time.time(), timeout)
        my_socket.close()
        return delay
def receive_ping(my_socket, packet_id, time_sent, timeout):
        # primeste ping de la soclu
        time_left = timeout
```

```
        while True:
            started_select = time.time()
            ready = select.select([my_socket], [], [], time_left)
            how_long_in_select = time.time() - started_select
            if ready[0] == []:    # timp limita pana crapa
                return
            time_received = time.time()
            rec_packet, addr = my_socket.recvfrom(1024)
            icmp_header = rec_packet[20:28]
            type, code, checksum, p_id, sequence = struct.unpack(
                'bbHHh', icmp_header)
            if p_id == packet_id:
                return time_received - time_sent
            time_left -= time_received - time_sent
            if time_left <= 0:
                return
def verbose_ping(dest_addr, timeout=2, count=4):
    # trimite un singur ping catre o singura destinatie (ip/nume gazda) si afiseaza
rezultatul
    for i in range(count):
        print('ping {}...'.format(dest_addr))
        delay = do_one_ping(dest_addr, timeout)
        if delay == None:
            print('(eroare nu a raspuns dupa {} secunde.)'.format(timeout))
        else:
            delay = round(delay * 1000.0, 4)
            print('am primit raspuns in {} milliseconds.'.format(delay))
    print('')
class PingQuery(asyncore.dispatcher):
    def __init__(self, host, p_id, timeout=0.5, ignore_errors=False):
        #clasa derivata din "asyncore.dispatcher"    pentru trimitere si receptare
icmp
        #de obicei aceasta clasa este utilizata impreuna cu loop din asyncore.
        # cand loop a incetat se pot primi rezultatele cu "get_result"
        asyncore.dispatcher.__init__(self)
        try:
            self.create_socket(socket.AF_INET, socket.SOCK_RAW, ICMP_CODE)
        except socket.error as e:
            if e.errno in ERROR_DESCR:
                # operatie nepermisa
                raise socket.error(''.join((e.args[1], ERROR_DESCR[e.errno])))
            raise    # generez eroarea originala
        self.time_received = 0
        self.time_sent = 0
        self.timeout = timeout
        self.packet_id = int((id(timeout) / p_id) % 65535)
        self.host = host
        self.packet = create_packet(self.packet_id)
        if ignore_errors:
            # ma fac ca ploua.
```

```
                self.handle_error = self.do_not_handle_errors
                self.handle_expt = self.do_not_handle_errors
        def writable(self):
                return self.time_sent == 0
        def handle_write(self):
                self.time_sent = time.time()
                while self.packet:
                        #utilizare port fals
                        sent = self.sendto(self.packet, (self.host, 1))
                        self.packet = self.packet[sent:]
        def readable(self):
                # cat timp nu trimitem nimic canalul trebuie sa ramana deschis
                if (not self.writable()
                                # odata incepta trimiterea trebuie sa mai verificam daca nu am
intarzieri in raspuns
                                and self.timeout < (time.time() - self.time_sent)):
                        self.close()
                        return False
                return not self.writable()
        def handle_read(self):
                read_time = time.time()
                packet, addr = self.recvfrom(1024)
                header = packet[20:28]
                type, code, checksum, p_id, sequence = struct.unpack("bbHHh", header)
                if p_id == self.packet_id:
                        self.time_received = read_time
                        self.close()
        def get_result(self):
                #intoarce intarzaierea la comunicare sau nimic
                if self.time_received > 0:
                        return self.time_received - self.time_sent
        def get_host(self):
                return self.host
        def do_not_handle_errors(self):
                pass
        def create_socket(self, family, type, proto):
                # rescris pentru a suporta si argumentul proto
                sock = socket.socket(family, type, proto)
                sock.setblocking(0)
                self.set_socket(sock)
                self.family_and_type = family, type
        def handle_connect(self):
                pass
        def handle_accept(self):
                pass
        def handle_close(self):
                self.close()

if __name__ == '__main__':
        # Testare
```

```
    verbose_ping('google.com')
    verbose_ping('127.0.0.1')
```

Interceptare de pachete

Acum că s-au mai clarificat unele lucruri putem trece la un exemplu care realizează un pic de intercepție și analiză de pachete - evident trebuie analizat cu documentația de tcp v4/v6 în față. NU uitați va rog că trebuie drepturi de admin deci din terminalul proiectului pycharm veti da *sudo python3 nume_fisier.py* pentru testare.

```
import socket
import struct
import textwrap
import binascii
import struct
import sys

TAB_1 = '\t - '
TAB_2 = '\t\t - '
TAB_3 = '\t\t\t - '
TAB_4 = '\t\t\t\t - '

DATA_TAB_1 = '\t     '
DATA_TAB_2 = '\t\t     '
DATA_TAB_3 = '\t\t\t     '
DATA_TAB_4 = '\t\t\t\t     '

def main():
    conn = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
    filters = (["ICMP", 1, "ICMPv6"],["UDP", 17, "UDP"], ["TCP", 6, "TCP"])
    filter = []

    if len(sys.argv) == 2:
        print("acesta este filtrul: ", sys.argv[1])
        for f in filters:
            if sys .argv[1] == f[0]:
                filter = f
    while True:
        raw_data, addr = conn.recvfrom(65536)
        dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)
        if eth_proto == 'IPV6':
            newPacket, nextProto = ipv6Header(data, filter)
            printPacketsV6(filter, nextProto, newPacket)
        elif eth_proto == 'IPV4':
            printPacketsV4(filter, data, raw_data)
def printPacketsV4(filter, data, raw_data):
    (version, header_length, ttl, proto, src, target, data) = ipv4_Packet(data)
    # ICMP
    if proto == 1 and (len(filter) == 0 or filter[1] == 1):
        icmp_type, code, checksum, data = icmp_packet(data)
        print ('**********************ICMP**********************')
```

```
            print ("\ttip ICMP: %s" % (icmp_type))
            print ("\tcod ICMP: %s" % (code))
            print ("\tsuma control ICMP: %s" % (checksum))
        # TCP
        elif proto == 6 and (len(filter) == 0 or filter[1] == 6):
            print('*****************TCPv4*********************')
            print('Versione: {}\nlungime header: {}\nTTL: {}'.format(version,
header_length, ttl))
            print('protocol: {}\nSursa: {}\nDestiinatia: {}'.format(proto, src, target))
            src_port, dest_port, sequence, acknowledgment, flag_urg, flag_ack,
flag_psh, flag_rst, flag_syn, flag_fin = struct.unpack(
                '! H H L L H H H H H H', raw_data[:24])
            print('*****Segmentul TCP*****')
            print('Portul sursei: {}\nDestination Port: {}'.format(src_port, dest_port))
            print('Secventa: {}\nAvizare: {}'.format(sequence, acknowledgment))
            print('*****Flag-uri*****')
            print('URG: {}\nACK: {}\nPSH: {}'.format(flag_urg, flag_ack, flag_psh))
            print('RST: {}\nSYN: {}\nFIN:{}'.format(flag_rst, flag_syn, flag_fin))
            if len(data) > 0:
                # HTTP
                if src_port == 80 or dest_port == 80:
                    print('*****date HTTP*****')
                    try:
                        http = HTTP(data)
                        http_info = str(http.data).split('\n')
                        for line in http_info:
                            print(str(line))
                    except:
                        print(format_output_line("",data))
                else:
                    print('*****date TCP*****')
                    print(format_output_line("",data))
        # UDP
        elif proto == 17 and (len(filter) == 0 or filter[1] == 17):
            print('*****************UDPv4*********************')
            print('Versiune: {}\nLungime header: {}\nTTL: {}'.format(version,
header_length, ttl))
            print('protocol: {}\nSursa: {}\nDestinatia: {}'.format(proto, src, target))
            src_port, dest_port, length, data = udp_seg(data)
            print('*****Segmentul UDP*****')
            print('Portul Sursei: {}\nPortul destinatiei: {}\nLength: {}'.format(src_port,
dest_port, length))
def printPacketsV6(filter, nextProto, newPacket):
    remainingPacket = ""
    if (nextProto == 'ICMPv6' and (len(filter) == 0 or filter[2] == "ICMPv6")):
        remainingPacket = icmpv6Header(newPacket)
    elif (nextProto == 'TCP' and (len(filter) == 0 or filter[2] == "TCP")):
        remainingPacket = tcpHeader(newPacket)
    elif (nextProto == 'UDP' and (len(filter) == 0 or filter[2] == "UDP")):
        remainingPacket = udpHeader(newPacket)
```

```
        return remainingPacket
def tcpHeader(newPacket):
        # 2 unsigned short,2 unsigned Int,4 unsigned short.
2byt+2byt+4byt+4byt+2byt+2byt+2byt+2byt==20byts
        packet = struct.unpack("!2H2I4H", newPacket[0:20])
        srcPort = packet[0]
        dstPort = packet[1]
        sqncNum = packet[2]
        acknNum = packet[3]
        dataOffset = packet[4] >> 12
        reserved = (packet[4] >> 6) & 0x003F
        tcpFlags = packet[4] & 0x003F
        urgFlag = tcpFlags & 0x0020
        ackFlag = tcpFlags & 0x0010
        pushFlag = tcpFlags & 0x0008
        resetFlag = tcpFlags & 0x0004
        synFlag = tcpFlags & 0x0002
        finFlag = tcpFlags & 0x0001
        window = packet[5]
        checkSum = packet[6]
        urgPntr = packet[7]
        print ("**********TCP**********")
        print ("\tPortul sursei: "+str(srcPort) )
        print ("\tPortul destinatiei: "+str(dstPort) )
        print ("\tNumarul secventei: "+str(sqncNum) )
        print ("\tNumarul validarii: "+str(acknNum) )
        print ("\tData Offset: "+str(dataOffset) )
        print ("\tRezervat: "+str(reserved) )
        print ("\tFlag-uri TCP: "+str(tcpFlags) )
        if(urgFlag == 32):
                print ("\tFlag Urgent: Activ")
        if(ackFlag == 16):
                print ("\tFlag Ack: Activ")
        if(pushFlag == 8):
                print ("\tFlag-ul Push: Activ")
        if(resetFlag == 4):
                print ("\tFlag-ul Reset: Activ")
        if(synFlag == 2):
                print ("\tFlag-ul Syn: Activ")
        if(finFlag == True):
                print ("\tFlag-ul Fin: Activ")
        print ("\tFereastra: "+str(window))
        print ("\tSuma de control: "+str(checkSum))
        print ("\tUrgent Pointer: "+str(urgPntr))
        print (" ")
        packet = packet[20:]
        return packet
def udpHeader(newPacket):
        packet = struct.unpack("!4H", newPacket[0:8])
```

```
        srcPort = packet[0]
        dstPort = packet[1]
        lenght = packet[2]
        checkSum = packet[3]
        print ("*****************UDP********************")
        print ("\tPortul sursei: "+str(srcPort))
        print ("\tPortul destinatiei: "+str(dstPort))
        print ("\tLungime: "+str(lenght))
        print ("\tSuma de control: "+str(checkSum))
        print (" ")
        packet = packet[8:]
        return packet
def icmpv6Header(data):
        ipv6_icmp_type, ipv6_icmp_code, ipv6_icmp_chekcsum = struct.unpack(
                ">BBH", data[:4])
        print ("*****************ICMPv6********************")
        print ("\tTip ICMPv6: %s" % (ipv6_icmp_type))
        print ("\tCod ICMPv6: %s" % (ipv6_icmp_code))
        print ("\tSuma de control ICMPv6: %s" % (ipv6_icmp_chekcsum))

        data = data[4:]
        return data
def nextHeader(ipv6_next_header):
        if (ipv6_next_header == 6):
                ipv6_next_header = 'TCP'
        elif (ipv6_next_header == 17):
                ipv6_next_header = 'UDP'
        elif (ipv6_next_header == 43):
                ipv6_next_header = 'Rutare'
        elif (ipv6_next_header == 1):
                ipv6_next_header = 'ICMP'
        elif (ipv6_next_header == 58):
                ipv6_next_header = 'ICMPv6'
        elif (ipv6_next_header == 44):
                ipv6_next_header = 'Fragment'
        elif (ipv6_next_header == 0):
                ipv6_next_header = 'HOPOPT'
        elif (ipv6_next_header == 60):
                ipv6_next_header = 'Destinatie'
        elif (ipv6_next_header == 51):
                ipv6_next_header = 'Autentificare'
        elif (ipv6_next_header == 50):
                ipv6_next_header = 'Incapsulare'
        return ipv6_next_header
def ipv6Header(data, filter):
        ipv6_first_word, ipv6_payload_legth, ipv6_next_header, ipv6_hoplimit =
struct.unpack(
                ">IHBB", data[0:8])
        ipv6_src_ip = socket.inet_ntop(socket.AF_INET6, data[8:24])
        ipv6_dst_ip = socket.inet_ntop(socket.AF_INET6, data[24:40])
```

```
        bin(ipv6_first_word)
        "{0:b}".format(ipv6_first_word)
        version = ipv6_first_word >> 28
        traffic_class = ipv6_first_word >> 16
        traffic_class = int(traffic_class) & 4095
        flow_label = int(ipv6_first_word) & 65535
        ipv6_next_header = nextHeader(ipv6_next_header)
        data = data[40:]
        return data, ipv6_next_header
# despachetare cadru Ethernet
def ethernet_frame(data):
        proto = ""
        IpHeader = struct.unpack("!6s6sH",data[0:14])
        dstMac = binascii.hexlify(IpHeader[0])
        srcMac = binascii.hexlify(IpHeader[1])
        protoType = IpHeader[2]
        nextProto = hex(protoType)
        if (nextProto == '0x800'):
                proto = 'IPV4'
        elif (nextProto == '0x86dd'):
                proto = 'IPV6'
        data = data[14:]
        return dstMac, srcMac, proto, data
        # Formatarea adresei MAC
def get_mac_addr(bytes_addr):
        bytes_str = map('{:02x}'.format, bytes_addr)
        mac_addr = ':'.join(bytes_str).upper()
        return mac_addr
# Despachetarea pachetelor IPv4 receptionate
def ipv4_Packet(data):
        version_header_len = data[0]
        version = version_header_len >> 4
        header_len = (version_header_len & 15) * 4
        ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
        return version, header_len, ttl, proto, ipv4(src), ipv4(target), data[header_len:]
# intoare o adresa de IP formatata
def ipv4(addr):
        return '.'.join(map(str, addr))
# despacheteaza orice pachet ICMP
def icmp_packet(data):
        icmp_type, code, checksum = struct.unpack('! B B H', data[:4])
        return icmp_type, code, checksum, data[4:]
# impacheteaza orice pachet tcp
def tcp_seg(data):
        (src_port, dest_port, sequence, acknowledgement, offset_reserved_flag) =
struct.unpack('! H H L L H', data[:14])
        offset = (offset_reserved_flag >> 12) * 4
        flag_urg = (offset_reserved_flag & 32) >> 5
        flag_ack = (offset_reserved_flag & 32) >> 4
        flag_psh = (offset_reserved_flag & 32) >> 3
```

```
        flag_rst = (offset_reserved_flag & 32) >> 2
        flag_syn = (offset_reserved_flag & 32) >> 1
        flag_fin = (offset_reserved_flag & 32) >> 1
        return src_port, dest_port, sequence, acknowledgement, flag_urg, flag_ack,
flag_psh, flag_rst, flag_syn, flag_fin, data[offset:]
# despacheteaza orice pachet UDP
def udp_seg(data):
        src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
        return src_port, dest_port, size, data[8:]
# Formatarea liniei de iesire
def format_output_line(prefix, string):
        size=80
        size -= len(prefix)
        if isinstance(string, bytes):
                string = ''.join(r'\x{:02x}'.format(byte) for byte in string)
                if size % 2:
                        size-= 1
                        return '\n'.join([prefix + line for line in textwrap.wrap(string, size)])
main()
```

Tema

Dacă ați înțeles aceste programe de acum pe baza lor puteți face orice fel de combinații doriți. De exemplu un proxy care face analiza pachetelor primite și eventual le trimite modificate mai departe sau chiar nu le trimite. Dacă mai săpați o să vedeți că o serie de abordări moderne din programare care sunt declarate sigure sunt extrem de sensibile la astfel de analize! Limbaje în care se poate lucra pentru temă - golang,rust,ruby,cpp,c,java,.net