

네트워크보안 과제 1

2019203021

소프트웨어학부

이승헌

## I. echo server

### 결과

```
root@29759742ea24: /homework1/task1
$ docker restart cs3
Error response from daemon: No such container: cs3

이승현 @DESKTOP-CJQ8192 MINGW64 ~/homework1 (main)
$ docker restart cs
Error response from daemon: No such container: cs

이승현 @DESKTOP-CJQ8192 MINGW64 ~/homework1 (main)
$ docker restart 297
297

이승현 @DESKTOP-CJQ8192 MINGW64 ~/homework1 (main)
$ bash run_docker.sh
root@29759742ea24: /homework1# ls
Dockerfile  build_docker.sh  rm_docker.sh  task1  task3
README.md   net75.out        run_docker.sh  task2  task4
root@29759742ea24: /homework1# cd task1
root@29759742ea24: /homework1/task1# ls
README.md  echo_client.py  echo_server.py
root@29759742ea24: /homework1/task1# python3 echo_server.py
Connected from ('127.0.0.1', 56050)
this is test
wow incredible

이승현 @DESKTOP-CJQ8192 MINGW64 ~
$ cd homework1/bash run_docker.sh
bash: cd: too many arguments

이승현 @DESKTOP-CJQ8192 MINGW64 ~
$ cd homework1; bash run_docker.sh
root@29759742ea24: /homework1# cd task1
root@29759742ea24: /homework1/task1# python3 echo_client.py
Traceback (most recent call last):
  File "/homework1/task1/echo_client.py", line 9, in <module>
    socket.connect((IP, PORT))
ConnectionRefusedError: [Errno 111] Connection refused
root@29759742ea24: /homework1/task1# python3 echo_client.py
this is test
this is test
wow incredible
wow incredible
```

## 코드

### 클라이언트

```
root@29759742ea24: /homework1/task1
import socket

IP = '127.0.0.1'
PORT = 9999
BUF_SIZE = 4096

socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#TODO: connect to the server
socket.connect((IP, PORT))

try:
    while True:
        your_input = input().encode()
        #TODO: send your input string to the server
        socket.send(your_input)
        response = socket.recv(BUF_SIZE)
        if not response: break
        print(response.decode())
except:
    socket.close()

~
```

### 서버

```
root@29759742ea24: /homework1/task1
import socket

IP = '127.0.0.1'
PORT = 9999
BUF_SIZE = 4096

#TODO: make a server-socket
server=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
#TODO: bind the IP, port number to the server-socket
server.bind((IP, PORT))
#TODO: make the socket a listening state
server.listen()
client, addr = server.accept()
print(f"Connected from {addr}")

try:
    while True:
        response = client.recv(BUF_SIZE)
        if not response: break
        #TODO: send the response back to the client
        buf=response.decode()
        client.send(response)
        print(buf)
except:
    client.close()
    server.close()

~
```

### 1. 작동 원리를 간단하게 설명하라.

서버가 로컬 호스트 주소의 포트 9999 번으로 오는 TCP 소켓 연결을 받도록 하고, 연결이 있을 때까지 대기한다. (서버의 `listen()`과 `accept()` 부분)

클라이언트는 로컬 호스트 주소의 포트 9999 번으로 `connect()` 메서드를 통해 TCP 소켓 연결을 한다. (이 때 서버는 켜져 있어야 한다)

클라이언트는 콘솔 창을 통해 입력한 문자열을 바이너리로 인코딩해서 서버로 전송한다. (`send`) 그리고 서버는 `recv` 메서드로 무언가를 받을 때까지 대기하고, 만약 받았다면 받은 것을 디코딩하여 출력한다. 그리고 받은 것을 그대로 클라이언트 쪽에 `send` 메서드로 보낸다. 이 시점까지 클라이언트는 `recv` 메서드에 의해 뭔가를 받을 때까지 대기하고 받았으면 디코딩해서 출력한다.

### 2. 8번째 줄(`server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)`)의 의미는 무엇인가?

테스트를 해보기 위해 해당 줄을 주석 처리하고 다음과 같이 행동했다.

- i) 서버에 클라이언트가 접속한다.
- ii) 클라이언트가 접속 중일 때 서버를 종료하고 그 즉시 다시 서버를 작동한다.

그 결과 포트가 이미 쓰이고 있다는 예외가 발생했다.

사용이 끝난 포트는 그 즉시 닫히는 것이 맞지만, 기본 동작이 지정된 시간만큼 `TIME_WAIT` 하는 것이어서 즉시 사용할 수가 없었다.

그 문제를 해결하기 위해서 `setsockopt`를 통해 `SO_REUSEADDR` 옵션을 설정해서 그 즉시 포트가 닫히도록 할 수 있다.

### 3. UDP 방식을 쓰려면 어디를 수정해야 하는가?

클라이언트와 서버의 `socket.socket()`에서 `socket.SOCK_STREAM`을 `socket.SOCK_DGRAM`으로 교체한다.

## II. Port scanner

### 결과

```
root@29759742ea24: /homework1/task2
root@29759742ea24:/homework1/task1# clear
root@29759742ea24:/homework1/task2# python3 port_scanner.py 10.0.100.3 10 100
open port:21
open port:22
open port:80
```

### 코드

```
root@29759742ea24: /homework1/task2
import socket
import sys

def port_scanner(target_ip, start_portno, end_portno):
    for port_number in range(start_portno, end_portno):
        #TODO: your code here
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        try:
            #TODO: your code here
            result=s.connect_ex((target_ip,port_number))
            s.close()
            #print(f"result:{result}")
            if result==0:
                print(f"open port:{port_number}")
        except ConnectionRefusedError:
            continue
        except TimeoutError:
            continue

if __name__ == '__main__':
    target_ip = sys.argv[1]
    start_portno = int(sys.argv[2])
    end_portno = int(sys.argv[3])

    port_scanner(target_ip, start_portno, end_portno)

~
~
```

## 1. 스캐너의 작동 방식

Connect\_ex() 메서드는 연결되면 0을, 연결이 되지 않았다면 그 이유에 맞는 에러 코드를 반환하는 C 스타일로 작성된 메서드다. 그래서 connect\_ex 메서드를 이용해 TCP 연결을 시도한다.

```
29759742ea24.42466 > task2.cs3864_net.http: Flags [S], cksum 0xdc33 (incorrect -> 0x9ffb), seq 3342397832, win 64240, options [mss 1460,sackOK,TS val 1099575440 ecr 0,nop,wscale 7], length 0
03:27:18.446464 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)
task2.cs3864_net.http > 29759742ea24.42466: Flags [S.], cksum 0xdc33 (incorrect -> 0xa598), seq 1051369603, ack 3342397833, win 65160, options [mss 1460,sackOK,TS val 1080416038 ecr 1099575440,nop,wscale 7], length 0
03:27:18.446470 IP (tos 0x0, ttl 64, id 31561, offset 0, flags [DF], proto TCP (6), length 52)
29759742ea24.42466 > task2.cs3864_net.http: Flags [.], cksum 0xdc2b (incorrect -> 0xd0f7), ack 1, win 502, options [nop,nop,TS val 1099575440 ecr 1080416038], length 0
03:27:18.446479 IP (tos 0x0, ttl 64, id 31562, offset 0, flags [DF], proto TCP (6), length 52)
29759742ea24.42466 > task2.cs3864_net.http: Flags [F.], cksum 0xdc2b (incorrect -> 0xd0f6), seq 1, ack 1, win 502, options [nop,nop,TS val 1099575440 ecr 1080416038], length 0
```

클라이언트에서 SYN을 보내고, 서버는 SYNACK를 보내며, 클라이언트는 ACK를 보내고 그 직후 바로 FIN을 보내서 끝맺는다.

이렇게 정상적인 TCP 3way handshake가 발생하면 0을 반환한다.

0을 반환한 포트의 경우 출력하게 된다.

```
03:27:18.446632 IP (tos 0x0, ttl 64, id 44426, offset 0, flags [DF], proto TCP (6), length 60)
29759742ea24.58120 > task2.cs3864_net.81: Flags [S], cksum 0xdc33 (incorrect -> 0x0134), seq 1526710114, win 64240, options [mss 1460,sackOK,TS val 1099575440 ecr 0,nop,wscale 7], length 0
03:27:18.446650 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
task2.cs3864_net.81 > 29759742ea24.58120: Flags [R.], cksum 0xd60e (correct), seq 0, ack 1526710115, win 0, length 0
```

클라이언트에서 SYN을 보냈지만 서버는 RST로 응답한다.

이 경우 그에 맞는 에러 코드를 반환하게 된다.

## 2. UDP 스캐닝을 하려면 어디를 수정해야하는가?

s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)을

s=socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)로 수정한다.

찾아본 방법에 의하면 소켓의 타임아웃을 settimeout() 메서드로 짧게 설정하고 sendto() 메서드와 recvfrom() 메서드를 사용하여 recvfrom으로 타임아웃 전에 뭔가를 받아오면 포트가 열려있고, 아니면 닫혀있다고 생각할 수 있다고 한다.

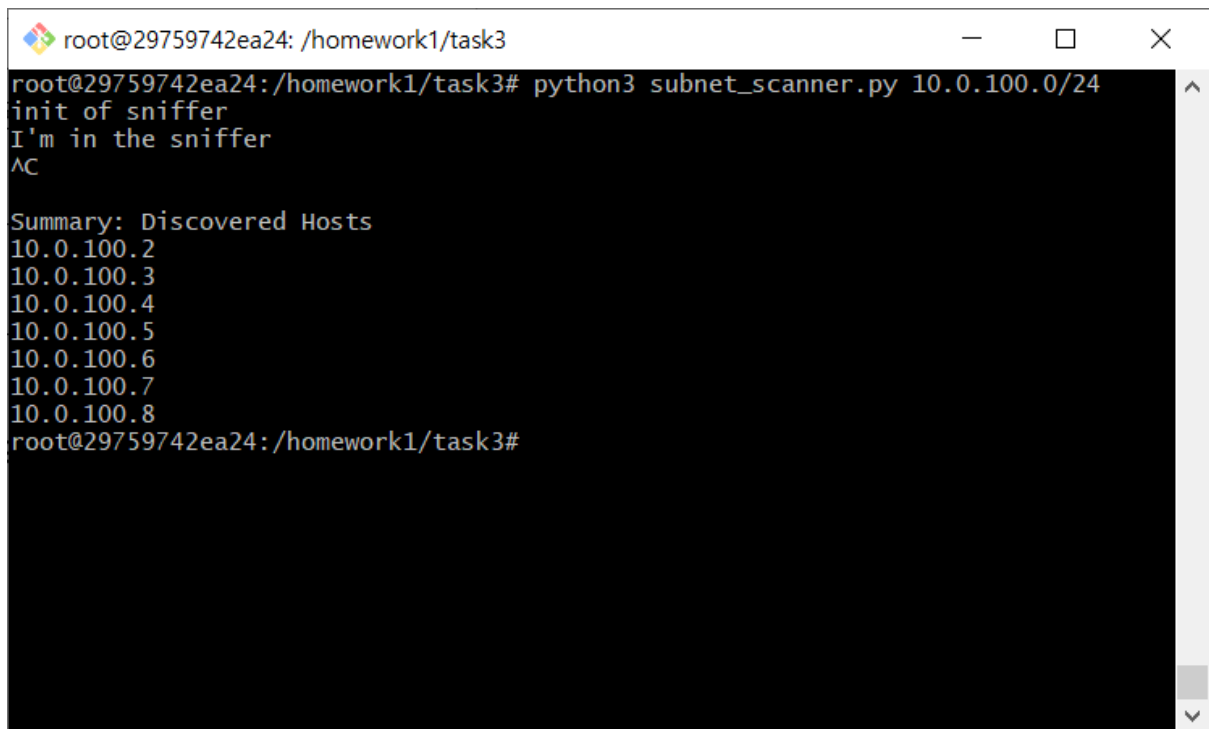
## 3. 닫힌 포트에 UDP 패킷을 보내면 UDP로 답할까? 아니라면 무엇으로 답할까?

다른 프로토콜을 이용해서 답을 보내는 것도 이상하다고 생각하여 UDP로 보낸다고 생각했다. 하지만 tcpdump로 관찰한 결과, 그리고 UDP 스캐닝에 대한 검색 결과 서버는 ICMP 메시지를 보내는 것을 알 수 있었다. 개인적인 이유를 생각해보자면 이렇다.

UDP 헤더는 주고받기 위한 정보 뿐이므로 포트가 닫힌 상태임을 알려려면 페이로드가 필요할 것이다. 즉, 알아서 페이로드를 만들어야 한다는 것인데 굳이 페이로드를 까야 하고, 통일되지 않은 방법으로 페이로드가 만들어진다면 코드로 이걸 하나하나 분석하기도 쉽지 않을 것이기 때문이라고 생각한다.

### 3. subnet scanner

결과



```
root@29759742ea24: /homework1/task3
root@29759742ea24:/homework1/task3# python3 subnet_scanner.py 10.0.100.0/24
init of sniffer
I'm in the sniffer
AC

Summary: Discovered Hosts
10.0.100.2
10.0.100.3
10.0.100.4
10.0.100.5
10.0.100.6
10.0.100.7
10.0.100.8
root@29759742ea24:/homework1/task3#
```

## 코드

```
root@29759742ea24: /homework1/task3
import socket
import ipaddress
import struct
import sys
import time
import threading

def change splitted_ip_array_to_decimal(arr_ip):
    return int(arr_ip[3])+(int(arr_ip[2])<<8)+(int(arr_ip[1])<<16)+(int(arr_ip[0])<<24)

def change_decimal_to_ip(deciaml_ip):
    return str((deciaml_ip & 0xff000000)>>24)+ "." +str(((deciaml_ip&0x00ff0000)>>16)+ "." +str((deciaml_ip & 0x0000ff00)>>8)+ "." +str(deciaml_ip& 0x000000ff))
```

ipaddress를 써도 되는 것을 파악하지 못했을 때 짠 메서드로,  
위의 메서드는 ip 문자열을 .로 split한 배열을 이용해 10진수로 ip를 변환하며,  
아래의 메서드는 10진수 ip를 다시 문자열로 만든다.

(중간은 IP, ICMP 클래스이므로 생략)

```
# sniffer: used to capture all ICMP packets come to this host.
def sniffer():
    print("init of sniffer")
    s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
    s.bind(("0.0.0.0", 0))
    s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

    discovered_hosts = set([])
    print("I'm in the sniffer")
    try:
        while True:
            raw_buffer = s.recvfrom(1500)[0]
            #TODO: your code here
            ip=IP(raw_buffer[:20])
            #print(f"ver:{ip.ver}, ihl:{ip.ihl}, tos:{ip.tos}, len:{ip.len}, id:{ip.id}, offset:{ip.offset},
            if ip.protocol=="ICMP":
                icmp=ICMP(raw_buffer[20:28])
                #print(f"type:{icmp.type}, code:{icmp.code}, sum:{icmp.sum}, id:{icmp.id}, seq:{icmp.seq}")
                if icmp.type==3 and icmp.code ==3:
                    discovered_hosts.add(ip.src_address)
    except KeyboardInterrupt:
        print(f'\n\nSummary: Discovered Hosts')
        for host in sorted(discovered_hosts):
            print(f'{host}')
        sys.exit()
```

raw\_buffer로 들어오는 데이터를 보니 40바이트가 넘어서 끊어서 써야 했다.  
들어오는 데이터가 어디서부터 IP인지 모르므로 IP는 20바이트, ICMP는  
8바이트라는 것만 알고 실험적으로 잘라본 결과 저렇게 자르면 알맞게 데이터가  
들어왔다.



```

# udp_sender: used to send UDP packets to all the hosts of a given subnet.
def udp_sender(subnet):
    STRING="SCAN"
    PORT=19999
    #TODO: your code here
    splitted_subnet=subnet.split('/')
    splitted_ip=splitted_subnet[0].split('.')
    mask=0xffffffff<<(32-int(splitted_subnet[1]))
    decimal_ip=change_splitted_ip_array_to_decimal(splitted_ip)
    decimal_subnet=decimal_ip&mask
    str_subnet=change_decimal_to_ip(decimal_ip)
    amount_of_host=0xffffffff>>int(splitted_subnet[1])
    list_of_host=list()
    for i in range(1,amount_of_host):
        host_ip=decimal_subnet+i
        str_ip=change_decimal_to_ip(host_ip)
        list_of_host.append(str_ip)

    s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    idx=0
    #for dst_ip in ipaddress.IPv4Network(subnet):
    for dst_ip in list_of_host:
        #print(f"ipaddr dst_ip:{dst_ip}")
        #print(f"list dst_ip:{list_of_host[idx]}")
        #idx=idx+1
        try:
            s.sendto(STRING.encode(),(str(dst_ip),PORT))
        except PermissionError:
            pass
if __name__ == '__main__':
    subnet = sys.argv[1]
    time.sleep(3)

    # execute a udp sender thread
    t = threading.Thread(target=udp_sender, args=(subnet,))
    t.start()

    # start sniffing
    sniffer()

```

위에 구현한 메서드를 이용해 직접 서브넷 마스크를 구하는 코드다.  
 /로 잘라서 ip와 cidr을 분리한다. 그리고 ip를 .로 잘라서 반환된 배열을  
 10진수로 변환하고, cidr을 이용해 마스크를 생성한다.  
 10진수 ip에 생성한 마스크를 씌워서 서브넷의 10진수 ip를 만들고,  
 그 10진수 서브넷 ip를 다시 문자열 ip로 변환한다.  
 호스트 개수는  $2^{(32 - \text{cidr})} - 2$  로 하여 호스트의 list에 위에서 만든 서브넷  
 ip를 이용하여 호스트의 ip를 넣는다.  
 위에서 만든 호스트 list를 활용해 각 호스트에 UDP 패킷을 19999번 포트로  
 보낸다.

## 1. 서브넷 스캐너의 작동 방식

Cidr으로 표현된 서브넷을 입력 받아서

다른 스레드에게는 입력을 토대로 서브넷의 모든 호스트를 알아내어

UDP 패킷을 보내도록 한다.

메인 스레드는 보낸 UDP 패킷에 의해 응답으로 오는 IP, ICMP패킷을 받게 한다.

메인 스레드는 종료 전까지 계속 실행되면서 원하는 패킷을 보내온 호스트를 기록한다.

두 과정이 절차적으로 이뤄지면 기껏 패킷을 다 보내도 응답을 잡아낼 수가 없기 때문에 스레드를 사용하는 것이다.

## 2. 서브넷 스캐닝은 UDP와 TCP 어떤 방식이 더 좋은가?

서브넷 아래에 있는 실질적인 호스트의 개수는  $2^{(32 - CIDR)} - 2$  개라고 생각할 수 있다. CIDR이 24 정도만 되어도 스캔할 호스트는 254개에 달한다.

이런 상황에서 3 way handshake를 해야만 하는 TCP는 오래 걸린다.

하지만 UDP는 보내기만 하면 서버로부터 다른 패킷을 기다릴 필요도 없이 끝이고, ICMP가 오기를 기다리기만 하면 된다.

따라서 서브넷 스캐닝은 UDP가 더 유리하다고 생각한다.

## 4. NIDS

### 결과

```
root@29759742ea24: /homework1/task4
I'm in the sniffer
AC
Summary: Discovered Hosts
10.0.100.2
10.0.100.3
10.0.100.4
10.0.100.5
10.0.100.6
10.0.100.7
10.0.100.8
root@29759742ea24:/homework1/task3# cd ../task4
root@29759742ea24:/homework1/task4# ls
README.md nids.py test.rules tester.py
root@29759742ea24:/homework1/task4# python3 nids.py
Traceback (most recent call last):
  File "/homework1/task4/nids.py", line 172, in <module>
    rule_file = sys.argv[1]
IndexError: list index out of range
root@29759742ea24:/homework1/task4# python3 nids.py test.rules
Start sniffing
2024.03.28 - 04:43:08 r1 packet for subnet 192.168.1.0 tcp 10.0.100.2 20 -> 192.168.1.100 264
2024.03.28 - 04:43:08 r1 packet for subnet 192.168.1.0 tcp 10.0.100.2 20 -> 192.168.1.100 692
2024.03.28 - 04:43:08 r2 packet for Telnet, FTP, and SSH tcp 127.0.0.1 20 -> 127.0.0.1 23
2024.03.28 - 04:43:08 r2 packet for Telnet, FTP, and SSH tcp 127.0.0.1 20 -> 127.0.0.1 25
2024.03.28 - 04:43:08 r2 packet for Telnet, FTP, and SSH tcp 127.0.0.1 20 -> 127.0.0.1 25
2024.03.28 - 04:43:09 r3 udp ports from 10000 to 20000 udp 127.0.0.1 53 -> 127.0.0.1 10400
2024.03.28 - 04:43:09 r3 udp ports from 10000 to 20000 udp 127.0.0.1 53 -> 127.0.0.1 12646
2024.03.28 - 04:43:09 r4 tcp SYN packet tcp 127.0.0.1 20 -> 127.0.0.1 80
2024.03.28 - 04:43:09 r5 HTTP GET message tcp 127.0.0.1 20 -> 127.0.0.1 80
2024.03.28 - 04:43:09 r6 remote shell execution tcp 127.0.0.1 20 -> 127.0.0.1 22
2024.03.28 - 04:43:09 r7 DNS query for Google open resolver udp 10.0.100.2 53 -> 8.8.8.8 53
2024.03.28 - 04:43:09 r8 ping to KWU icmp 10.0.100.2 0 -> 223.194.1.180 0
2024.03.28 - 04:43:09 r9 ping icmp 127.0.0.1 0 -> 127.0.0.1 0
```

코드

nids.py

(윗줄은 Rule 클래스이므로 생략)

root@29759742ea24: /homework1/task4

```
def parse_rule(line):
    #TODO: your code here
    line_without_enter=line.replace("\n", "")
    line_without_last=line_without_enter[:-2]
    header,body=line_without_last.split("(")

    src_infos=str()
    dst_infos=str()
    direction=str()
    if("<" in header):
        src_infos,dst_infos=header.split("<")
        direction="<"
    else:
        src_infos,dst_infos=header.split("->")
        direction="->"

    src_components=src_infos.strip().split(" ")
    dst_components=dst_infos.strip().split(" ")
    components_of_body=body.strip().split(";")

    action=src_components[0]
    protocol=src_components[1]
    src_ip=src_components[2]
    src_port=src_components[3]
    dst_ip=dst_components[0]
    dst_port=dst_components[1]

    msg=str()
    options=list()
    if protocol in option_dict.keys():
        options=option_dict[protocol].copy()
    custom_rule=list()
    for component in components_of_body:
        component_no_blank=component.strip()
        key, value = component_no_blank.split(":")
        if component_no_blank.startswith("msg"):
            msg=value.replace("'", "")
        else:
            idx=0
            for option in options:
                if key == option:
                    options[idx]=value
                    break
            idx=idx+1
        else:
            custom_rule.append(component_no_blank)
    return Rule(action,protocol,src_ip,src_port,direction,dst_ip,dst_port,options,msg,custom_rule)
```

요구사항에 있는 snort 형식의 텍스트가 온다고 가정한다면 잘 작동하는 rule 파싱 함수다.

Rule 클래스에 필요한 파라미터를 뽑아 낼 수 있도록 문자열을 전처리한 후 split하는 것이 코드의 전부라고 할 수 있다.

상단의 option\_dict가 options의 기준이라고 생각했기에 options에 존재하지 않거나 페이로드에 무엇이 있냐를 물어보는 조건인 경우 original\_rule에 할당되도록 했다.

root@29759742ea24: /homework1/task4

```
def parse_packet(packet, rule_set):
    #TODO: your code here

    ip_packet=packet[IP]
    src_ip=ip_packet.src
    dst_ip=ip_packet.dst
    protocol=packet.proto
    src_port=int()
    dst_port=int()
    itype=int()
    icode=int()
    payload=None
    flag=str()
    if(packet.haslayer(TCP)==True):
        src_port=packet[TCP].sport
        dst_port=packet[TCP].dport
        flag=packet[TCP].flags
    if(packet.haslayer(UDP)==True):
        src_port=packet[UDP].sport
        dst_port=packet[UDP].dport
    if(packet.haslayer(ICMP)==True):
        itype=packet[ICMP].type
        icode=packet[ICMP].code

    if(packet.haslayer(Raw)):
        payload=packet[Raw].load.decode()

    mapped_rule=None

    for rule in rule_set:
        if(rule.src_ip!="any"):
            for ip in ipaddress.IPv4Network(rule.src_ip):
                if(str(ip)==str(src_ip)):
                    break
            else:
                continue
        if(rule.dst_ip!="any"):
            for ip in ipaddress.IPv4Network(rule.dst_ip):
                if(str(ip)==str(dst_ip)):
                    break
            else:
                continue
        if(rule.protocol != protocol_dict[protocol]):
            continue
        elif(rule.protocol == "tcp" or rule.protocol=="udp"):
            if(rule.protocol=="tcp" and rule.dst_port=="any" and rule.options[3]!="flags" and rule.options[3] != flag):
                continue

        src_target_list=list()
        src_start=int()
        src_end=int()

        if(rule.src_port!="any" and len(rule.src_port.split(",")==1):
            range_of_src_port=rule.src_port.split(":")
            if(len(range_of_src_port)==1):
                src_start=int(range_of_src_port[0])
```

root@29759742ea24: /homework1/task4

```
        stc_end=start
    else:
        src_start=int(range_of_src_port[0])
        src_end=int(range_of_src_port[1])
        if(src_port < src_start or src_port > src_end):
            continue
    elif(rule.src_port!="any" and len(rule.src_port.split(","))>1):
        src_target_list=rule.src_port.split(",")
        idx=0
        for str_target in src_target_list:
            src_target_list[idx]=int(str_target)
            idx=idx+1
        if(src_port not in src_target_list):
            continue

    dst_target_list=list()
    dst_start=int()
    dst_end=int()
    if(rule.dst_port!="any" and len(rule.dst_port.split(","))==1):
        range_of_dst_port=rule.dst_port.split(":")
        if(len(range_of_dst_port)==1):
            dst_start=int(range_of_dst_port[0])
            dst_end=dst_start
        else:
            dst_start=int(range_of_dst_port[0])
            dst_end=int(range_of_dst_port[1])
        if(dst_port < dst_start or dst_port > dst_end):
            continue
    elif(rule.dst_port!="any" and len(rule.dst_port.split(","))>1):
        dst_target_list=rule.dst_port.split(",")
        idx=0
        for str_target in dst_target_list:
            dst_target_list[idx]=int(str_target)
            idx=idx+1
        if(dst_port not in dst_target_list):
            continue
    elif(rule.protocol == "icmp"):
        if(itype!=int(rule.options[0]) and icode!=int(rule.options[1])):
            continue

    if(payload is not None):
        if(len(rule.original_rule)==0):
            continue
        for custom_rule in rule.original_rule:
            key,value = custom_rule.strip().split(":")
            key=key.strip()
            value=value.strip().replace("'",'')
            if(key=="content" and (value in payload or value == payload)):
                break;
        else:
            continue

    mapped_rule=rule
    break

if(mapped_rule is not None):
    print(f"{datetime.now().strftime('%Y.%m.%d - %H:%M:%S')} {mapped_rule.msg} {mapped_rule.protocol} {src_ip} {src_port} {mapped_rule.direction} {dst_ip} {dst_port}")
```

rule\_set에 있는 모든 rule을 반복문을 통해서 패킷과 대조하는데, rule의 모든 필드와 패킷의 각 필드에 대응되는 정보를 if문으로 비교하여 모두 통과한 경우 해당 rule을 mapped\_rule에 할당하고 반복문을 빠져나온다.  
그리고 mapped\_rule이 None이 아니면 요구사항의 형식에 맞게 출력한다.

## tester.py

```
root@29759742ea24: /homework1/task4
# Tester example. Build your own tester to verify your NIDS!
from scapy.all import *
import random

if __name__ == '__main__':
    r1_1 = (Ether() / IP(dst = "192.168.1.100") / TCP(dport = random.randrange(1, 1024)))
    sendp(r1_1, iface='eth0')
    r1_2 = (Ether() / IP(dst="192.168.1.100") / TCP(dport=random.randrange(1,1024)))
    sendp(r1_2, iface='eth0')

    r2_1=(Ether()/IP()/TCP(dport=23))
    r2_2=(Ether()/IP()/TCP(dport=25))
    r2_3=(Ether()/IP()/TCP(dport=21))
    sendp(r2_1,iface='eth0')
    sendp(r2_2,iface='eth0')
    sendp(r2_3,iface='eth0')

    r3_1=(Ether()/IP()/UDP(dport=random.randrange(10000,20000)))
    r3_2=(Ether()/IP()/UDP(dport=random.randrange(10000,20000)))
    sendp(r3_1,iface='eth0')
    sendp(r3_2,iface='eth0')

    r4_1=(Ether()/IP()/TCP(Flags="S"))
    sendp(r4_1,iface='eth0')

    r5_1=(Ether()/IP()/TCP(dport=80)/"GET")
    sendp(r5_1,iface="eth0")

    r6_1=(Ether()/IP()/TCP(dport=22)/"/bin/sh")
    sendp(r6_1,iface="eth0")

    r7_1=(Ether()/IP(dst="8.8.8.8")/UDP(dport=53))
    sendp(r7_1,iface="eth0")

    r8_1=(Ether()/IP(dst="223.194.1.180")/ICMP())
    sendp(r8_1,iface="eth0")

    r9_1=(Ether()/IP()/ICMP())
    sendp(r9_1,iface="eth0")
```

요구사항에 맞는 패킷을 보낸다.

임의의 포트나 지정된 여러 개 포트가 조건인 경우 여러 번 보냈고, 그 외에 경우는 한 번만 보냈다.

### 1. 작동 방식과 구현 방법

입력된 snort 형식으로 적은 text 파일을 읽어서 각 line을 strip(), replace()을 이용하여 전처리하고 split()을 이용해 필요한 정보를 추출하여 Rule 객체로 만들고 각 rule을 rule\_set에 저장한다.

그리고 rule\_set을 이용하여 감지한 패킷을 rule\_set에 있는 rule과 대조한다. 패킷의 정보를 추출하는 과정에서 사용하는 if문은 해당 프로토콜이 패킷에 존재하는지 묻는 것으로, 해당하는 프로토콜이 없는데 해당 프로토콜의 정보를 추출하려고 하면 예외가 발생하여 추가하게 되었다.

rule 객체의 각각의 필드에 매핑되는 패킷의 정보를 if문으로 같은가 다른가를 판단하여 대조한다. 모든 if문에 걸리지 않고 통과한 경우 rule에 일치하는 패킷이라고 판단하게 된다.

### 2. 효율적으로 구현하기 위해 사용한 방법

기능의 작동에 충실하게 구현하였기 때문에 효율적으로 구현하였다고 보기에는 어렵다.

### 3. 대규모 네트워크의 이 NIDS에 사용할 수 있는가?

절대로 사용할 수 없을 것이라고 생각한다.

추가된 Rule이 많아지면 많아질수록, Rule의 필드가 늘어나거나, protocol이 추가되거나 options의 비교 개수가 늘어나는 등 Rule의 경우의 수가 많아지면 코드의 유지보수가 힘든 것은 물론, if문이 더 많아질 것이므로 성능도 더 나빠지리라고 예상한다.