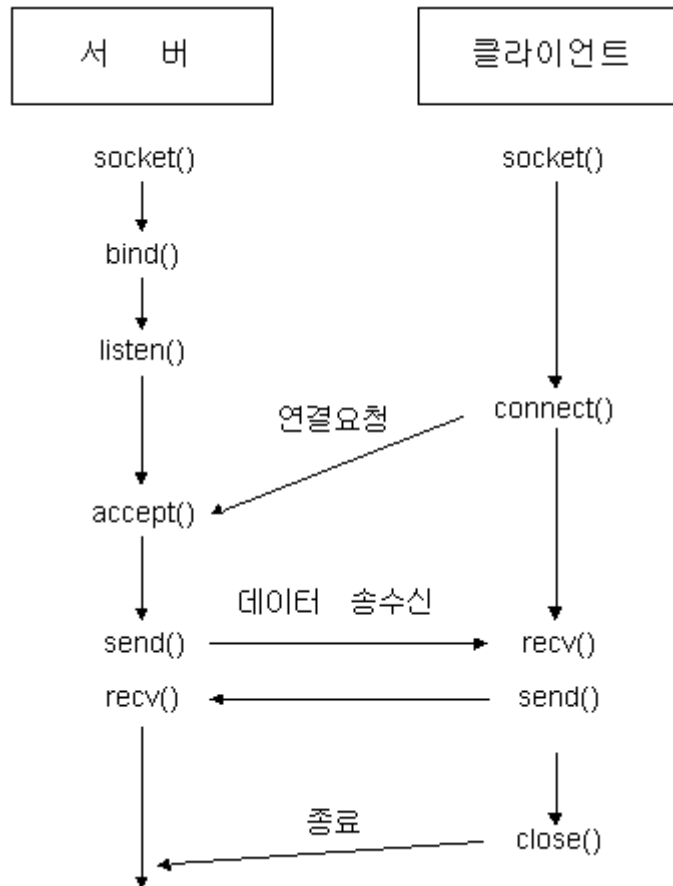


1. TASK 1

>>1-1. 프로그램 작동방식



-----서버에서-----

Socket() - 소켓 만들기

= like 전화기 만들기

Bind() - ip와 port를 묶어 socket bound. 인터페이스 결합

= like 내 전화번호 설정

Listen() - 연결을 받아들이기 위한 대기. 백로그큐 설정

= like 전화기 개통

Accept() - 클라이언트로부터 connect가 들어오면 받아들임 with three-way handshake

= like 전화가 오면 받음

Recv() - 클라이언트가 보낸 데이터를 읽음(바이트 오브젝트), 연결이 끝나면 b"를 받음

Send() – 데이터를 보냄

-----클라이언트에서-----

Socket() – 소켓 만들기

= like 전화기 만들기

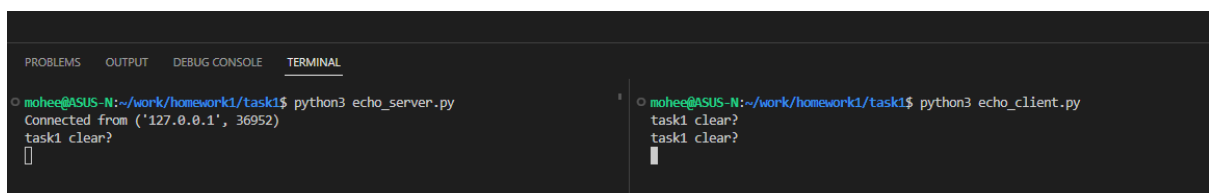
Connect(target) – 해당 서버로 연결

= like 전화걸기

Recv() – 서버가 보낸 데이터를 읽음(바이트 오브젝트)

Send() – 데이터를 보냄

Close() – 연결 해제



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
mohee@ASUS-N:~/work/homework1/task1$ python3 echo_server.py
Connected from ('127.0.0.1', 36952)
task1 clear?
[]
mohee@ASUS-N:~/work/homework1/task1$ python3 echo_client.py
task1 clear?
task1 clear?
```

Echo 결과. client에서 보낸 메시지를 서버에서 recv로 읽고, 다시 받은 메시지를 send해서 클라이언트에서 recv로 받아낸 결과.

>>1-2. echo_server.py line 8 의 목적

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

AF_INET = Internet Protocol v4 주소(소켓이 통신 할 수 있는 주소 유형).

SOCK_STREAM = 해당 소켓에 TCP 패킷을 보낼 것이라는 의미.

>>1-3. TCP 대신 UDP를 사용하려면?

Socket.**SOCK_STREAM** 대신 Socket.**SOCK_DGRAM** 사용

2. TASK 2

>>2-1. 스캐너 작동 방식

```
def port_scanner(target_ip, start_portno, end_portno):  
  
    for port_number in range(start_portno, end_portno):  
        try:  
            ip = target_ip  
            scanner_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
            scanner_socket.connect((ip, port_number))  
            scanner_socket.send(b'testing\n')  
            result = scanner_socket.recv(4096)  
            print('%d TCP OPEN'% port_number)
```

target_ip에 대해 start포트번호부터, end포트번호까지 반복:

try: //중간에 오류발생시 break

소켓생성

Target_ip와 port_number에 connect 시도

연결되면 아무 데이터 'testing' send

Recv()로 데이터 받기

Print(오픈되어있는 TCP 포트번호)

타겟 ip의 지정한 포트번호들에게 connect 요청을 보내 확인함.

----Connect()에서----

해당 포트가 open, closed 상태일 때 모두 syn+ack 반환.

하지만 closed일땐, client와 연결이 완성되지 않아 더 이상 실행되지않음.

따라서 open인 경우에만, print(포트번호)까지 실행됨.

```
root@e6f1f75aa013:/homework1/task2# python3 port_scanner.py 10.0.100.3 1 100
21 TCP OPEN
22 TCP OPEN
80 TCP OPEN
root@e6f1f75aa013:/homework1/task2#
```

예제를 통해 10.0.100.3의 포트번호 1부터 100부터 port scanning을 한 결과

>>2-2. TCP 대신 UDP 스캐닝을 수행하려면?

Scanner_socket을 생성할 때, Socket.SOCK_STREAM 대신 Socket.SOCK_DGRAM 사용

>>2-3. 닫힌 포트에 UDP 패킷을 보낸다면 호스트가 어떤 패킷으로 응답할까?

열린포트에 보낼 시, UDP응답이 오거나 응답이 오지않음.

닫힌포트에 보낼시, ICMP메시지(Type 3:Destination Unreachable, Code 3:Port Unreachable) 응답.

3. TASK 3

>>3-1. 서브넷 스캐너 작동 방식

연결된 서브네트워크의 모든 호스트로 테스트 메시지가 포함된 ip데이터그램을 전송하고 응답을 분석함. 대부분의 ICMP 메시지는 라우터에서 생성되어 응답. Port Unreachable 메시지는 호스트가 생성하여 응답함.

따라서 수신되는 ICMP메시지에 Port Unreachable 메시지가 있다면 해당 호스트는 동작하고 있음을 의미함.

```
t = threading.Thread(target=udp_sender, args=(subnet,))
t.start()

# start sniffing
print('start sniffing')
sniffer()
```

스레드를 사용하여, udp_sender의 작동이 시작된 뒤, 작동되는 동안 sniffer()가 작동 되도록 함.

----udp_sender()에서----

```
udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
for ip in ipaddress.ip_network(subnet).hosts():

    try:
        udp_socket.sendto(STRING.encode('utf-8'), ('%s'%ip, PORT))
        print('SENDING to [%s]' %ip)

    except Exception as e:
        print(e)
```

udp_socket 생성후, 서브넷에있는 호스트들에게 test문자를 보냄.

----Sniffer()에서----

```
if protocol == 'ICMP':
    offset = ipheaderlen
    icmp = ICMP(buff = raw_buffer[offset:][:8])
    icmp_type = icmp.type
    icmp_code = icmp.code
    if icmp_type == 3 and icmp_code == 3:
```

```

        t1 = struct.unpack('!12s', raw_buffer[-12:])
        print('%s -> %s : ICMP: Type[%d], Code[%d]'%(src_ip,
dst_ip,icmp_type,icmp_code))
        discovered_hosts.add('%s -> %s : ICMP: Type[%d],
Code[%d]'%(src_ip, dst_ip,icmp_type,icmp_code))

```

받은 ICMP 메시지 타입이 3, 코드값이 3일 경우 udp_scender 스캐너가 보낸 메시지와 같은지 확인.

같다면 해당 호스트는 동작중임.

```

SENDING to [10.0.100.247]
SENDING to [10.0.100.248]
SENDING to [10.0.100.249]
SENDING to [10.0.100.250]
SENDING to [10.0.100.251]
SENDING to [10.0.100.252]
SENDING to [10.0.100.253]
SENDING to [10.0.100.254]
^C

Summary: Discovered Hosts
10.0.100.2 -> 10.0.100.2 : ICMP: Type[3], Code[3]
10.0.100.3 -> 10.0.100.2 : ICMP: Type[3], Code[3]
10.0.100.4 -> 10.0.100.2 : ICMP: Type[3], Code[3]
10.0.100.5 -> 10.0.100.2 : ICMP: Type[3], Code[3]
10.0.100.6 -> 10.0.100.2 : ICMP: Type[3], Code[3]
10.0.100.7 -> 10.0.100.2 : ICMP: Type[3], Code[3]
10.0.100.8 -> 10.0.100.2 : ICMP: Type[3], Code[3]
root@e6f1f75aa013:/homework1/task3# python3 subnet_scanner.py 10.0.100
.0/24

```

예제에서 10.0.100.0/24 서브넷을 스캔함으로써, 7개의 호스트를 확인.

>>3-2. TCP, UDP중 어떤 프로토콜이 가장 좋을까?

UDP scan은 ICMP Port Unreachable 메시지를 받으면 close 상태이며, 메시지가 오지 않으면 open 상태이다. 하지만, close 상태는 명확하게 포트가 닫혀 있다는 것을 알 수 있지만, open 상태는 UDP 비연결형 protocol 특성상 네트워크의 상태 등에 의해 응답이 없을 수 있기 때문에 open 상태에 대해서는 정확도가 떨어질 수 있다. 따라서 3-way handshake를 통하는 TCP가 더 좋다.

4. TASK 4

>>4-1. NIDS 작동방식과 구현방법

```
if __name__ == '__main__':
    rule_file = sys.argv[1]

    f = open(rule_file, 'r')

    rule_set = []
    lines = f.readlines()
    for line in lines:
        rule = parse_rule(line)
        rule_set.append(rule)

    # print(rule_set)

    print("Start sniffing")
    sniff(iface='eth0', prn=lambda p: parse_packet(p, rule_set), filter='ip')

    f.close()
```

rule이 저장된 파일을 불러옴.

텍스트를 한줄한줄 불러온뒤,

Parse_rule(line)을 통해 분석하여 Rule 정보를 얻은뒤,

Rule_set 리스트에 각각 저장.

----Sniff()에서----

Parse_packet을 통해, 패킷과 동일한 rule이있는지 rule_set안과 비교.

rule안의 특성과 맞는 패킷일 경우, 해당 rule의 msg를 출력하게됨.

----parse_rule(line)에서----

```
word = line.split(' ')

action = word[0]
protocol = word[1]
src_ip=word[2]
src_port=word[3]
direction=word[4]
```



```
dst_ip=word[5]
dst_port=word[6]
```

라인을 ' '기준으로 분리한뒤, 0~5번째 인덱스의 값은 다음과 같음.

```
# Options
strs = line.split('(')
if (len(strs) >= 2):
    # remove trailing ')' if present
    if (strs[-1][-1] == ')'):
        strs[-1] = strs[-1][: -1]

    # options may be present
    options = {'tos':None, 'len':None, 'offset':None, 'seq':None,
'ack':None, 'itype':None,\
               'icode':None, 'flags':None, 'http_request':None,
'content':None}
    opts = strs[1].split(';')
    for opt in opts:
        kv = opt.split(':',1)
        if (len(kv) >= 2):
            option = kv[0].strip()
            value = kv[1].strip()
            # print(option, value)

            if (option == "msg"):
                msg = value
            elif (option == "tos"):
                options['tos'] = int(value)
            elif (option == "len"):
                options['len'] = int(value)
            elif (option == "offset"):
                options['offset'] = int(value)
            elif (option == "seq"):
                options['seq'] = int(value)
            elif (option == "ack"):
                options['ack'] = int(value)

            ...

            elif (option == "content"):
                options['content'] = value
                # remove starting and ending ["]
                if (options['content'].endswith('')):
                    options['content'] = options['content'][: -1]
                if (options['content'].startswith('')):
                    options['content'] = options['content'][1:]
```

```

        else:
            raise ValueError("Invalid rule : incorrect option : '"
+ option + "'.")

```

이후 option필드를 구해줌. Option을 존재 가능한 필드들에 대해 딕셔너리 형태로 표현.

해당 필드값이 존재할 경우 해당 값을 해당 필드에 넣어줌.

```

rule = Rule(action=action, protocol=protocol, src_ip=src_ip,
            src_port=src_port, direction=direction, dst_ip=dst_ip,
            dst_port=dst_port, options = options, msg = msg,
            original_rule = line)
return rule

```

마지막으로 구한 값들을 이용해 Rule 객체 생성한 뒤 return

----parse_packet(packet, rule_set)에서----

```

for rule in rule_set:

    mached = True

```

rule_set 안에 있는 모든 rule을 반복해서 packet과 비교.

비교도중 맞지 않는 것이 나올 경우 mached 값은 False로 바뀜.

```

#check protocol
f = False
answer_protocol = ''
if rule.protocol == 'icmp' and ICMP in packet:
    f = True
    answer_protocol = 'icmp'
elif rule.protocol == 'tcp' and TCP in packet:
    f = True
    answer_protocol = 'tcp'
elif rule.protocol == 'udp' and UDP in packet:
    f = True
    answer_protocol = 'udp'
if not(f): mached = False

```

rule과 packet의 **protocol**이 서로 같은지 확인

```

#check Ip_src and destination
f= False
answer_src_ip = ''
answer_dst_ip = ''
if (IP not in packet):
    f = False
else:
    srcIP = packet[IP].src
    dstIP = packet[IP].dst
    ipSrc = ipaddress.ip_address(srcIP)
    ipDst = ipaddress.ip_address(dstIP)
    if (((str(ipSrc) in rule.src_ip) or rule.src_ip=='any') and
((str(ipDst) in rule.dst_ip)) or rule.dst_ip=='any')):
        f = True
        answer_src_ip = srcIP
        answer_dst_ip = dstIP
    else:
        f = False
if not(f): mached = False
# print('check ip-----', answer_src_ip, answer_dst_ip)

#check source Port
f= False
answer_src_port = ''
answer_dst_port = ''
if (UDP in packet):
    srcPort = packet[UDP].sport
    dstPort = packet[UDP].dport
    if (((str(srcPort) in rule.src_port) or rule.src_port=='any') and
((str(dstPort) in rule.dst_port) or rule.dst_port=='any')):
        f = True
        answer_src_port = srcPort
        answer_dst_port = dstPort
elif (TCP in packet):
    srcPort = packet[TCP].sport
    dstPort = packet[TCP].dport
    if (((str(srcPort) in rule.src_port) or rule.src_port=='any') and
((str(dstPort) in rule.dst_port) or rule.dst_port=='any')):
        f = True
        answer_src_port = srcPort
        answer_dst_port = dstPort
elif (ICMP in packet):
    srcPort = packet[ICMP].sport
    dstPort = packet[ICMP].dport
    if (((str(srcPort) in rule.src_port) or rule.src_port=='any') and
((str(dstPort) in rule.dst_port) or rule.dst_port=='any')):
        f = True

```

```

        answer_src_port = srcPort
        answer_dst_port = dstPort
    if not(f): mached = False

```

source ip, port 그리고 destination ip, port가 서로 같은지 확인

rule에서 any일경우 아무거나 와도 매칭됨으로 간주.

```

#check options
f = True
if (rule.options['tos'] is not None):
    if (IP in packet):
        if(rule.options['tos'] != int(packet[IP].tos)):
            f = False
    else : f = False
if (rule.options['len'] is not None):
    if (IP in packet):
        if(rule.options['len'] != int(packet[IP].ihl)):
            f = False
    else : f= False

    ...

if (rule.options['content'] is not None):
    payload = None
    if (TCP in packet):
        payload = packet[TCP].payload
    elif (UDP in packet):
        payload = packet[UDP].payload
    if (payload):
        if (rule.options['content'] not in str(payload)):
            f = False
    else:
        f = False
if not(f): matched = False

```

option이 서로 같은지 확인

```

if (mached):
    print(datetime.now().strftime('%c'),' ',rule.msg,'
',answer_protocol,' ',\
        answer_src_ip,' ',answer_src_port,' ','->','\
        answer_dst_ip,' ',answer_dst_port)

```

마지막 비교까지 다른게 없었다면, 즉, rule과 일치한다면, 현재 시간, 해당룰의msg, src_ip, src_port,

dst_ip, dst_port를 출력해줌.

False										21 TCP OPEN
Sun Apr 9 18:32:44 2023	"r4 tcp SYN packet"	tcp	10.0.100.2	55974	-> 10.0.100.3	61				22 TCP OPEN
Sun Apr 9 18:32:44 2023	"my task4 success!"	tcp	10.0.100.2	55974	-> 10.0.100.3	61				80 TCP OPEN
False										root@e6f1f75aa013:/homework1/task2# python3
Sun Apr 9 18:32:44 2023	"r4 tcp SYN packet"	tcp	10.0.100.2	57728	-> 10.0.100.3	62				port_scanner.py 10.0.100.3 1 100
Sun Apr 9 18:32:44 2023	"my task4 success!"	tcp	10.0.100.2	57728	-> 10.0.100.3	62				21 TCP OPEN
False										22 TCP OPEN
Sun Apr 9 18:32:44 2023	"r4 tcp SYN packet"	tcp	10.0.100.2	38774	-> 10.0.100.3	63				80 TCP OPEN
Sun Apr 9 18:32:44 2023	"my task4 success!"	tcp	10.0.100.2	38774	-> 10.0.100.3	63				root@e6f1f75aa013:/homework1/task2# python3
False										port_scanner.py 10.0.100.3 1 100
Sun Apr 9 18:32:44 2023	"r4 tcp SYN packet"	tcp	10.0.100.2	55860	-> 10.0.100.3	64				21 TCP OPEN
Sun Apr 9 18:32:44 2023	"my task4 success!"	tcp	10.0.100.2	55860	-> 10.0.100.3	64				22 TCP OPEN
False										80 TCP OPEN
Sun Apr 9 18:32:44 2023	"r4 tcp SYN packet"	tcp	10.0.100.2	52842	-> 10.0.100.3	77				root@e6f1f75aa013:/homework1/task2# python3
Sun Apr 9 18:32:44 2023	"my task4 success!"	tcp	10.0.100.2	52842	-> 10.0.100.3	77				port_scanner.py 10.0.100.3 1 100
Sun Apr 9 18:32:44 2023	"r4 tcp SYN packet"	tcp	10.0.100.2	57790	-> 10.0.100.3	84				21 TCP OPEN
Sun Apr 9 18:32:44 2023	"my task4 success!"	tcp	10.0.100.2	57790	-> 10.0.100.3	84				22 TCP OPEN
										80 TCP OPEN
										root@e6f1f75aa013:/homework1/task2#

Test.rules에 다음을 추가 한 뒤, task2의 포트 스캐너를 실행한 결과

```
alert tcp 10.0.100.2 any -> 10.0.100.3 any (msg:"my task4 success!";)
```

>>4-2. 효율적 구현

```
if not(f):  
    mached = False  
    continue
```

각 rule과 packet을 비교할 때, 앞부분부터 비교하면서 다른부분이 있을경우, 나머지 부분을 비교하지 않고 바로 다음 rule과 비교할 수 있도록함. -> rule 비교시간 단축

추가적으로, rule_set을 정의할 때, 리스트타입 말고, 조금더 참조 시간을 줄일 수 있는 구조를 고민해보았으나, 답을 찾지 못함.

>>4-3. 대규모 네트워크에 사용가능할까?

네트워크 또는 호스트에 영향을 최소화하지만, 트래픽 양에 영향을 받아 대규모 네트워크의 경우에는 성능이 떨어질 수 있다.

하지만, 적절하게 배치하면 어느정도 넓은 네트워크 감시가 가능하다.

<http://www.koreascience.or.kr/article/CFKO201021868481319.pdf>

참고 : 병렬 구조 NIDS를 위한 효율적인 플로우 기반 부하 분산 기법에 관한 연구