네트워크 보안

Homewrok1

김진우교수님

2019203069 이시은

Task 1

[server]

TCP 연결을 하기 위해서는 서버의 소켓과 클라이언트의 소켓이 필요하다. 이 부분이 비어있었고, 따라서 서버의 소켓을 만들어 주었다.

```
#TODO: make a server-socket
server= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

“socket.AF_INET” 은 IPv4 주소 체계를 사용한다는걸 의미하고, “socket.SOCK_STREAM”은 소켓의 타입 중 TCP 소켓을 사용할 것을 의미한다. 따라서 이 소켓 객체는 IPv4 주소 체계를 사용하는 TCP 소켓이다.

bind 를 통해 전에 생성했던 server 라는 소켓에 IP 와 PORT 번호를 지정해준다

```
#TODO: bind the IP, port number to the server-socket
server.bind((IP, PORT))

#TODO: make the socket a listening state
server.listen()
```

그런다음 server 소켓은 클라이언트의 연결 요청을 기다리며, 연결 요청이 오면 클라이언트 소켓과 연결되어 데이터를 주고받게 된다.

또한 클라이언트에게 다시 응답을 보내는 요구사항도 충족시켜주었다.

```
    #TODO: send the response back to the client
    client.send(response)
```

```
    print(buf.decode())    "buf" is not defined
```

원래 코드에서는 buf 이라는 변수가 정의가 되어있지 않아서 의아했다. 실행을 해보니 클라이언트에서 메시지를 보냈을 때 서버가 바로 꺼졌었고, 그래서 해당 부분을 지웠다

[client]

client 의 소켓에 어느 서버에 연결할지를 지정해준다.

```
#TODO: connect to the server
socket.connect((IP,PORT))
```

그런다음 연결된 서버에게 메시지를 보내야 했는데, send 를 이용해 클라이언트 쪽에서 입력받은 input 을 전달해주었다.

```
    #TODO: send your input string to the server
    socket.send(your_input)
```

Required Questions

1. Briefly describe how your program works.

tcp 소켓을 사용하여 연결지향 통신을 한다. 서버소켓은 주어진 IP 주소와 포트 번호에 바인딩되어 클라이언트의 연결을 수신하고, 클라이언트 소켓은 서버에 연결하여 메시지를 보낸다. 그 메세지는 서버쪽에서도 출력이 되고, 서버는 받은 메시지를 다시 클라이언트에게 되돌려준다.

2. What's the purpose of `line 8` code in the `echo_server.py`?

```
8       server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

소켓을 다시 바인딩할 때 이전에 사용된 포트를 재사용할 수 있도록 하여 서버를 중단시키지 않고도 포트를 다시 사용할 수 있게하기 위한 코드이다.

3. If you want to use UDP instead of TCP in this program, which code snippets should be modified?

통신 프로토콜을 이에 맞게 socket.socket(socket.AF_INET, socket.SOCK_STREAM) 에서 socket.SOCK_STREAM 부분을 socket.SOCK_DGRAM 으로 바꿔줘야 한다. 또한 TCP 와 다르게 UDP 는 연결지향적이 아니기 때문에 server.accept()를 할 필요도 없다.

Task 2

[port_scanner.py]

인자로 받은 IP 주소로부터, 다음 인자인 포트시작 번호와 끝번호의 범위만큼의 포트번호를 스캐닝한다. 스캔은 네트워크의 연결된 호스트를 보는것인데, TCP 연결에서는 TCP 를 사용하는 특정 port 의 상태가 open 인지 close 인지를 타겟이 사용중인 포트번호로 어느 서비스를 실행중인지 유추를 통해 알아낼수있다.

python3 port_scanner.py 192.168.0.1 1 100

이런식으로 실행해줘야한다. IP addrress 는 run_test_server.sh 를 실행시켜서 얻은 dummy server 의 IP 를 썼다.

또한 과제에서 connect()매서드 대신 connect_ex(): 를 이용했다. 전자로 했을 경우 연결이 잘 되면 소켓을 반환하지만 실패를 했을 경우 예외를 발생시켜 프로그램이 도중에 중단되어 원하는 결과를 얻을 수 없었다. 하지만 후자의 매서드를 이용하는 경우 연결이 잘 되었을 때 0 을 반환하고, 연결을 실패하여도 예외를 발생시키지 않는다. 오류 코드만 반환할 뿐 프로그램 중단 없이 실패한 연결을 처리할 수 있다. 따라서 다음포트로 계속해서 포트 스캐닝을 할 수 있다.

또 마주했던 에러가 있었는데, dummy container 로부터 연결은 잘 된듯 싶었지만, 계속해서 connect_ex 의 결과가 0 이 나오지 않았다. 확인해 보니 task2 컨테이너에서 apt 가 잘 동작하지 않았던 문제였고, apt 를 apt-get 으로 바꾸어주었더니 컨테이너가 제대로 만들어지고 service 도 옳게 start 되어서 만들었던 port_scanner 가 잘 동작할 수 있게 되었다.

```
#6 41.93   404  Not Found [IP: 185.125.190.36 80]
#6 41.97 Fetched 42.7 MB in 39s (1092 kB/s)
#6 41.97 E: Failed to fetch http://ports.ubuntu.com/ubuntu-ports/pool/main/c/curl/libcurl4_7.81.0-1ubuntu1.15_arm64.deb  404  Not Found [IP: 185.125.190.36 80]
#6 41.97 E: Unable to fetch some archives, maybe run apt-get update or try with --fix-missing?
------
executor failed running [/bin/sh -c apt install openssh-server vsftpd apache2 net-tools -y]: exit code: 100
04a10249281962b445d605f4294c150f6c7e701afbea7555570cd0f0920c0ccf
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "10.0.100.2",
```

```
# python3 port_scanner.py 10.0.100.2 0 256
Port 0's result: 111
Port 1's result: 111
Port 2's result: 111
Port 3's result: 111
Port 4's result: 111
Port 5's result: 111
Port 6's result: 111
Port 7's result: 111
Port 8's result: 111
Port 9's result: 111
Port 10's result: 111
Port 11's result: 111
Port 12's result: 111
Port 13's result: 111
Port 14's result: 111
Port 15's result: 111
Port 16's result: 111
Port 17's result: 111
Port 18's result: 111
Port 19's result: 111
Port 20's result: 111
Port 21's result: 111
Port 22's result: 111
```

task2 > Dockerfile > ...

```
1  FROM ubuntu:jammy
2  RUN apt-get update -y
3  RUN apt-get install openssh-server vsftpd apache2 net-tools -y
4  COPY entrypoint.sh /entrypoint.sh
5
6  ENTRYPOINT service ssh start && service vsftpd start && service apache2 start && bash
7
```

```
root@a6a887b86c68:/homework1/task2# python3 port_scanner.py 10.0.100
.2 0 256
Port 21 is open
Port 22 is open
Port 80 is open
```

Required Questions

1. Describe how your scanner works. Concretely, describe how your logic scans open/closed ports of a target host.
Connect_ex()를 이용하여 인자로 주어진 start_portno ~ end_portno 범위의 각 포트들에 대해 순차적으로 TCP 연결을 시도한다. 만약에 포트가 open 되어있다면 0 을 리턴한다. 만약 연결이 거부되거나 타임아웃이 발생했을 경우에는 예외를 처리하며 계속 진행한다.
이를 통해 열려있는 포트를 식별하여 출력할 수 있게 된다.

2. If you want to perform UDP scanning instead of TCP, which part of this script should be modified?
소켓을 생성할때 socket.socket(socket.AF_INET, socket.SOCK_DGRAM)로, UDP 타입으로 생성해야한다. 또한 UDP 스캐닝은 TCP 와 달리 연결 설정 과정이 없으므로 socket.connect_ex()대신 socket.sendto()함수를 이용하여 패킷을 보내야한다.

3. If you run tcpdump while performing this task, you will see that if a port is closed, an opposing host will reply with a TCP RST packet. Suppose you send a UDP packet to a closed port on a specific host. Do you think the host will reply with a UDP packet? If not, which packet are you likely to receive?
포트가 닫혀있는 경우 TCP 스캐닝에서는 대상 호스트가 TCP RST 패킷을 보내는 경우가 있는데, UDP 스캐닝은 연결 지향적이지 않으므로 응답을 잘 하지 않는다.

Task 3

처음에 8 줄이 출렸되었었는데 알고보니 task2 에서 가동시켰던 컨테이너였다. 이를 종료시키니 옳게 7 개로 출력되었다.

```
root@071066afd804:/homework1/task3# python3 subnet_scanner.py 10.0.100.0/24
^C
10.0.100.1
10.0.100.2
10.0.100.3
10.0.100.4
10.0.100.5
10.0.100.6
10.0.100.7
```

또한 기존의 코드에서처럼 변수 이름이 socket 으로 되어있었는데, 이는 기존에 내장되어있던 모듈과의 혼란을 주어서 제대로 실행이 되지 않았으므로 sock 으로 바꾸었다.

Required Questions

1. Describe how your subnet scanner works. You need to mention a *thread*.

Udp_sender 함수는 주어진 서브넷의 모든 호스트에 대해 upd 패킷을 보낸다. 이 함수에서는 스레드를 사용하여 각 호스트에 대한 전송을 병렬로 처리한다. 이렇게 하면 각 호스트에 동시에 udp 패킷을 보낼 수 있어 전체 스캔 시간이 단축되기 때문이다. 그런 다음 sniffer 함수가 icmp 패킷을 받고 icmp 응답을 받은 호스트를 발견한다. 이 두 함수가 동시에 실행되어 udp 패킷을 보내는 작업과 icmp 패킷을 받아 호스트를 발견하는 작업이 동시에 이루어질 수 있다.

2. In the case of subnet scanning, which protocol would be best between UDP and TCP? Explain why you think so.

icmp 프로토콜이 호스트의 상태를 확인하는데 가장 적절한 프로토콜이라 일반적으로 서브넷 스캐닝에 많이 쓰이지만 tcp 나 udp 는 각각 연결지향, 비연결지향 프로토콜이므로 서브넷 스캐닝에는 적합하지 않다. 연결 설정 및 해제할 때에 발생하는 시간이 꽤 걸리기 때문이다. 하지만 이는 속도적인 측면에서, TCP 보단 UDP 가 비연결 지향성을 추구하여 더 적은 리소스가 필요하기 때문에 더 우세하므로 UDP 가 적절해 보인다.

Task 4

[parse_rule]

line 단위로 읽어들이면서 파싱을 하는 구조였다. 라인을 띄어쓰기를 기준으로 parts 로 나누고나서 rule_body 부분을 다시 붙인 다음, 거기에서 msg 에 해당하는 부분을 추출했어야했다. 하지만 test.rules 에 나와있는 line 마다 msg 의 위치가 달라서 처음에 구현했을 땐 제대로 되지 않았었다. 그러다가 rule_body 를 ()를 제외 한 안쪽 부분을 가지고 ‘;’ 단위로 split 한 다음 msg 가 있는 부분을 따로 찾아서 이 문제를 해결할 수 있었다.

[parse_packet]

여기에서는 프로토콜 별로 패킷에서 ip 와 port 에 접근하는 방법이 달랐어서 처음에 tcp 에 초점을 두고 코드를 짰었는데, tester 로 여러 패킷에 대해서 시험해 보았을 때 프로토콜이 udp 나 icmp 인 것들은 제대로 동작하지 않았다. 그래서 이를 해결하기 위해 port 와 ip 를 체크하는 부분에서

```
if protocol == 'tcp' and TCP in packet:
    tcp = packet[TCP]
    src_port = str(tcp.sport)
    dst_port = str(tcp.dport)
elif protocol == 'udp' and UDP in packet:
    udp = packet[UDP]
    src_port = str(udp.sport)
    dst_port = str(udp.dport)
elif protocol == 'icmp' and ICMP in packet:
    icmp = packet[ICMP]
    src_port = 'N/A'
    dst_port = 'N/A'
else:
    return
```

위의 사진과 같이 각 프로토콜 방식에 맞게 ip 와 port 를 가져오게끔 구현하였다.

Required Questions

1. Describe how your NIDS work and how to implement it.

Test.rules 에 있는 rule 들을 각각 파싱을 하여 ruleset 에 저장을 한 다음, 들어오는 패킷을 분석하여 ruleset 과 일치하는 부분이 있는지 확인을 한다. 이렇게해서 일치하는 부분이 있다면 메세지를 과제에 주어진 형식에 맞게 출력을 한다.

2. What method did you use for implementing NIDS efficiently?

spacy 를 이용하여 패킷 캡쳐, 분석, 조작을 쉽게 수행할 수 있었다. ruleset 과 비교를 하기 위해서 패킷을 분석해야 했는데, 네트워크 인터페이스를 실시간으로 감시하여 새로운 패킷이 올 때마다 바로 분석 및 처리를 하는 비동기적인 방식을 사용하는 spacy 덕분에 쉽게 해결할 수 있었다.

3. Do you think that your NIDS can be used for a large network?

처리해야할 트래픽이 많아지면 ruleset 의 크기와 복잡성이 커져서 성능이 충분히 좋지 않아지기 때문에 대규모 네트워크에서 사용하기에는 무리가 있을것 같다.