

| NCHAT

소켓 통신을 이용한 채팅 프로그램 개발 |

서재형
2023_NIT

INDEX

1
네트워크란? - OSI 7 layer

2
Socket이란?

3
Socket의 종류

4
HTTP 통신 vs Socket 통신

5
Socket 통신 원리를 이용한
채팅 프로그램 개발

6
Wireshark를 이용한
채팅 프로그램 통신 패킷 분석

7
보안 취약점 분석 및 프로그램 개선 방안

개발 배경 및 목적

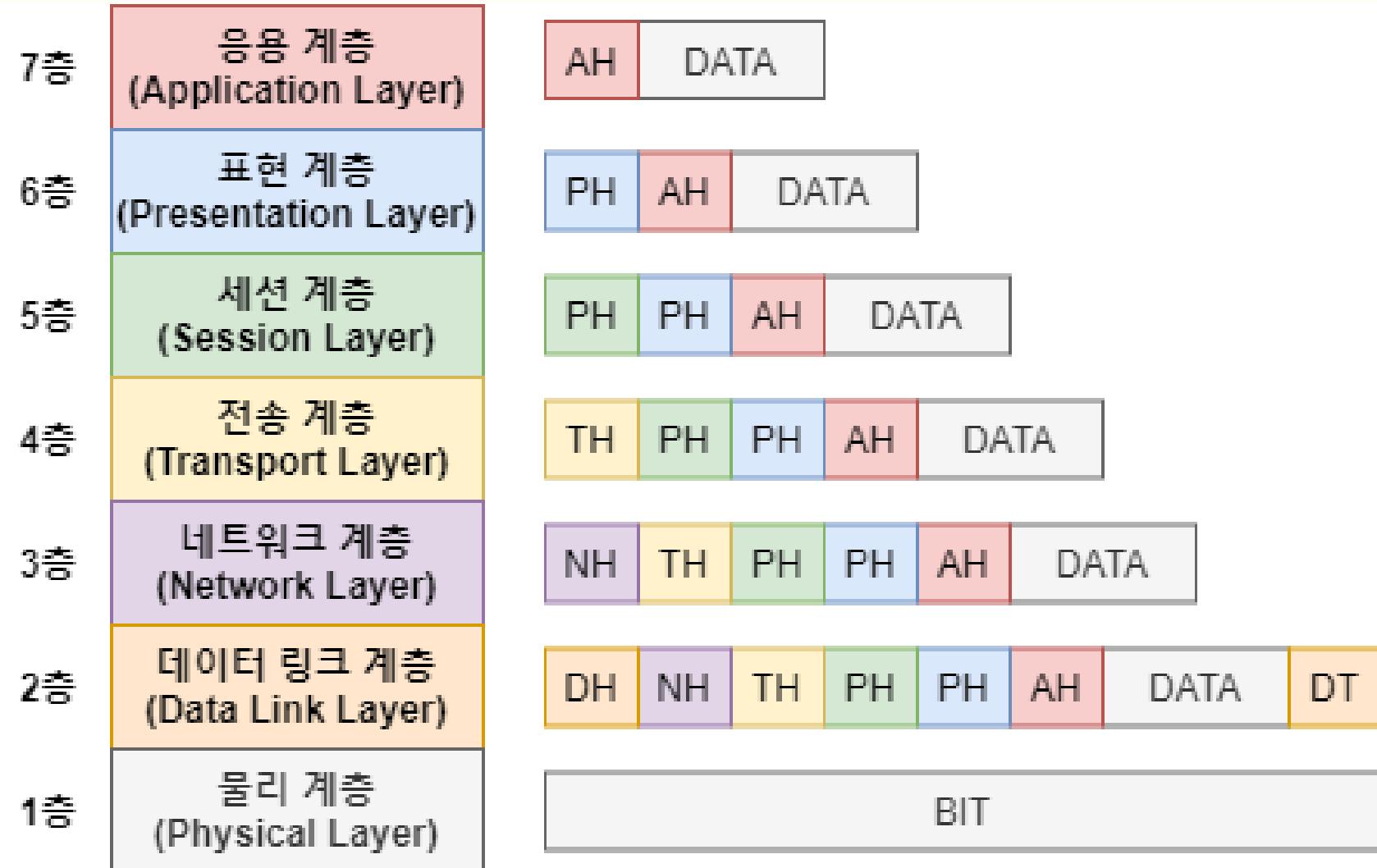
배경

- 새로운 측면의 주제 탐구
- 시큐어 코딩의 중요성 증가

목적

- 전반적인 네트워크 구조 및 통신 원리에 대한 이해
- Socket 통신에 대한 이해 및 활용
- GUI 코딩에 threading이 필요한 이유에 대한 이해
- 피드백 과정과 시큐어 코딩의 중요성 파악

네트워크란? - OSI 7 layer(TCP/IP 4 layer)



- **OSI 7 layer**

초기 여러 정보 통신 업체 장비들은 자신의 업체
장비들끼리만 연결이 되어 호환성이 없다는 문제가 존재

→ 모든 시스템들의 상호 연결에 있어
문제없도록 표준을 정한 것이 OSI 7계층

※실제 인터넷에서 사용되는 TCP/IP는 OSI 참조 모델을 기반
으로 상업적이고 실무적으로 이용될 수 있도록 단순화한 것임

네트워크란? - OSI 7 layer(TCP/IP 4 layer)

OSI 7 Layer

L7	응용 계층 (Application Layer)
L6	표현 계층 (Presentation Layer)
L5	세션 계층 (Session Layer)
L4	전송 계층 (Transport Layer)
L3	네트워크 계층 (Network Layer)
L2	데이터 링크 계층 (Data Link Layer)
L1	물리 계층 (Physical Layer)

TCP/IP 4 Layer

L4	응용 계층 (Application Layer)
L3	전송 계층 (Transport Layer)
L2	인터넷 계층 (Internet Layer)
L1	네트워크 액세스 (Network Access Layer)

- TCP/IP(UDP/IP) 4 layer

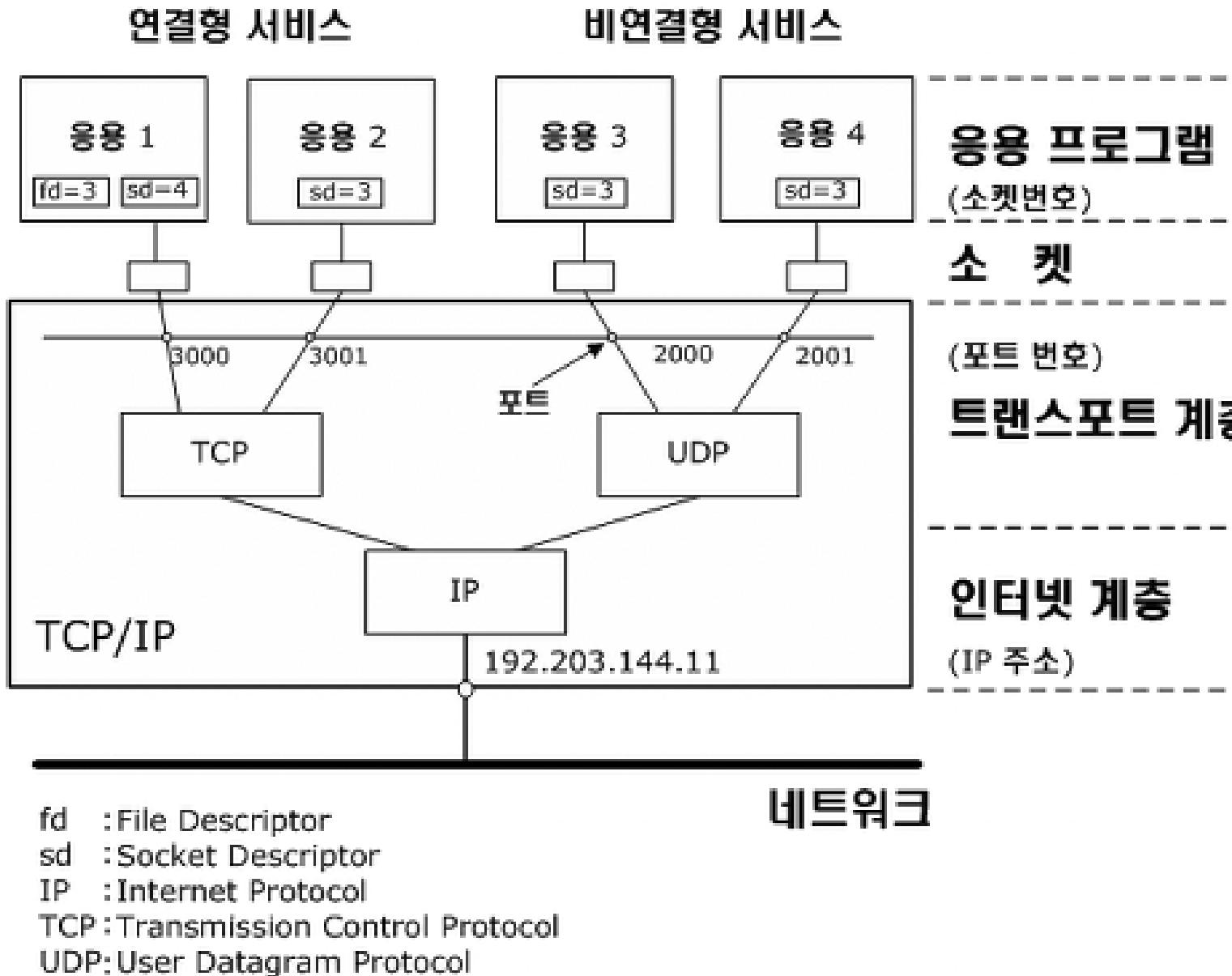
L4 응용 계층(Application Layer)

L3 전송 계층(Transport Layer)

L2 인터넷 계층(Internet Layer)

L1 네트워크 연결 계층(Network Access Layer)

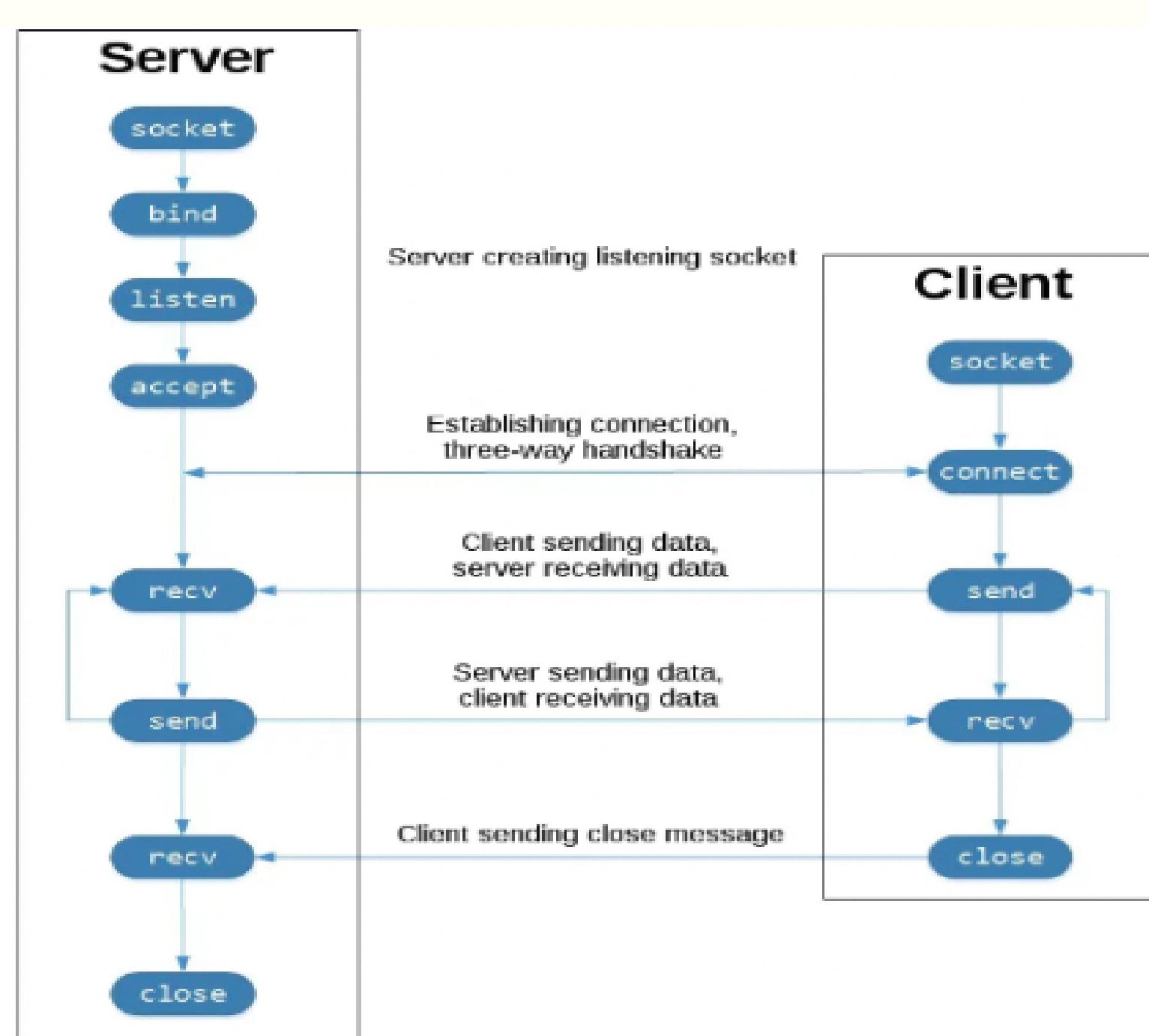
Socket이란?



Socket

- 네트워크를 경유하는 **프로세스 간 통신의 종착점**
- 응용 계층에 속하는 **프로세스들은 데이터 송수신을 위해 반드시 소켓을 거쳐 전송 계층으로 데이터를 전달해야 함**
- 소켓은 전송 계층과 **프로그램 사이의 인터페이스 역할을 함**

Socket의 종류

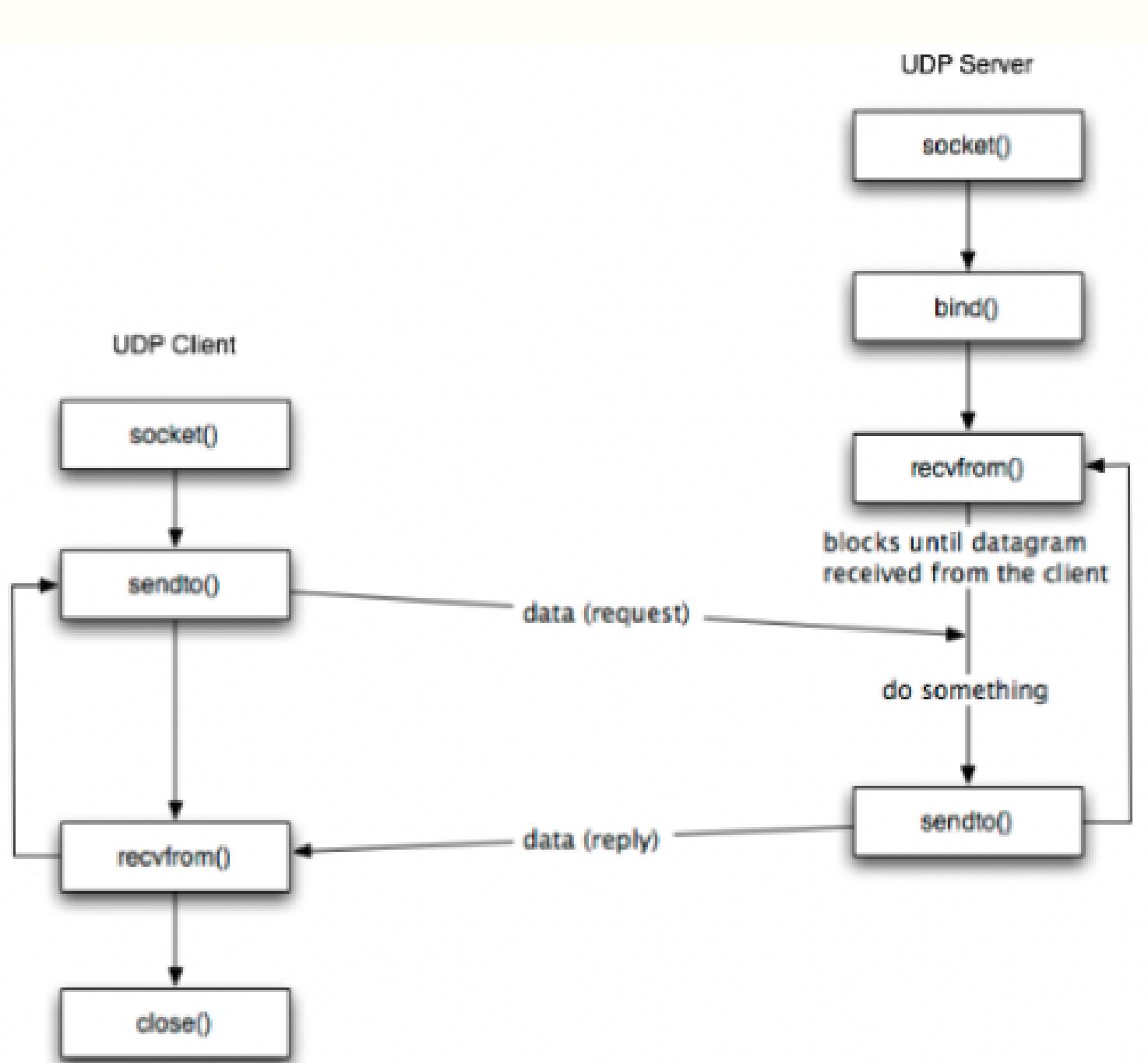


스트림 소켓

"TCP(Transmission Control Protocol)을
사용하는 연결 지향방식의 소켓"

- 송수신자의 연결을 보장하여 신뢰성 있는 데이터 송수신이 가능하다.
 - 데이터의 순서를 보장한다. (신뢰성 상승)
 - 소량의 데이터보다 대량의 데이터 전송에 적합하다.
 - 점대점 연결

Socket의 종류



데이터그램 소켓

"UDP(User Datagram Protocol)을 사용하는 비연결형 소켓"

- 데이터의 순서와 신뢰성을 보장하기 어렵다.
- 절대점 뿐만 아니라 일대다 연결도 가능하다.

HTTP 통신 vs Socket 통신

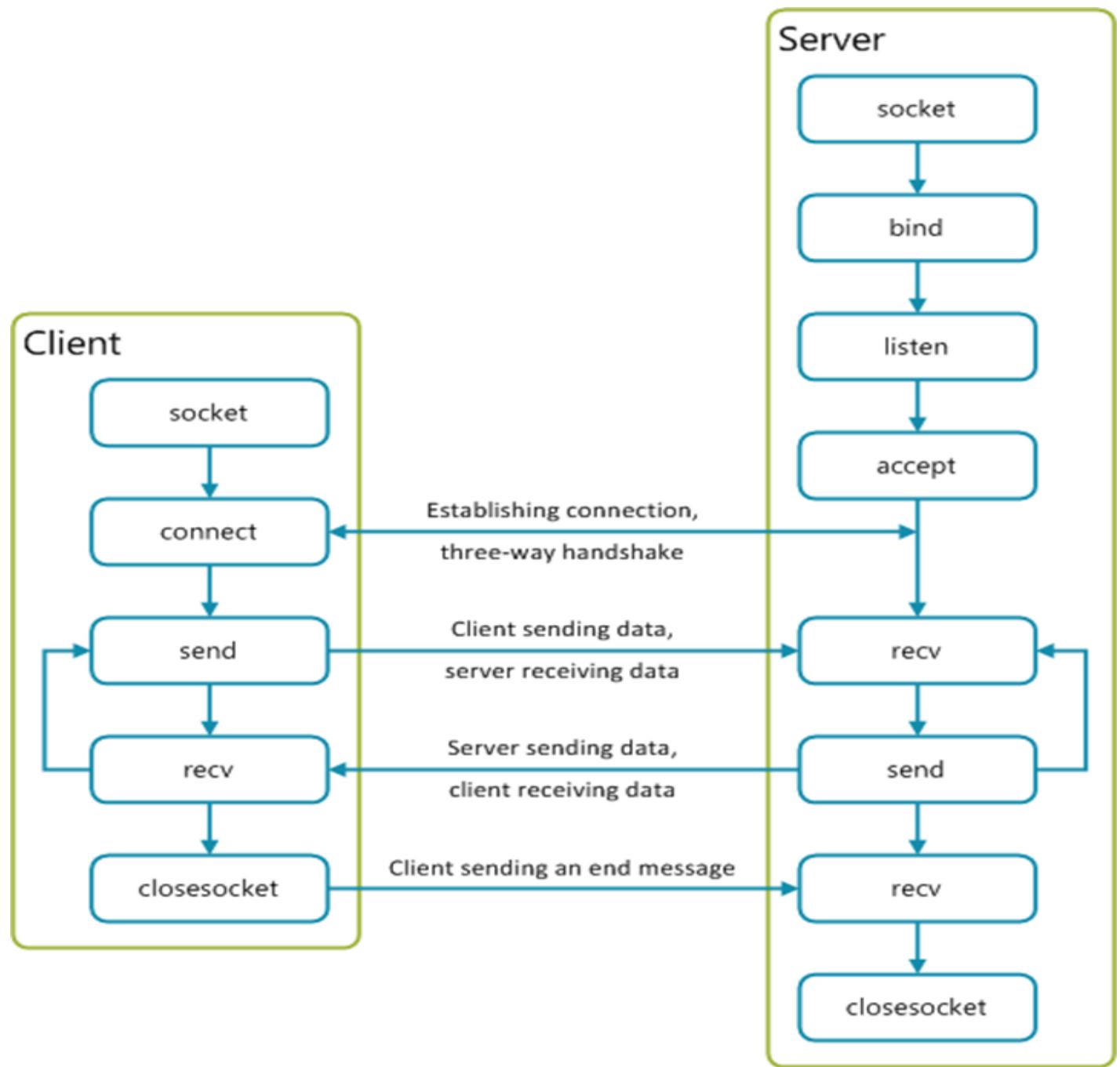
HTTP 통신

- 클라이언트의 요청이 있을 때만 서버가 응답
- JSON, HTML, Image 등 다양한 데이터를 주고 받을 수 있음
- 서버가 응답한 후 연결을 바로 종료하는 단방향 통신이지만 Keep Alive 옵션을 주어 일정 시간동안 커넥션을 유지할 수 있다.
- 실시간 연결이 아닌 데이터 전달이 필요한 경우에만 요청을 보내는 상황에 유리

Socket 통신

- 클라이언트와 서버가 특정 포트를 통해 양방향 통신을 하는 방식
- 데이터 전달 후 연결이 끊어지는 것이 아니라 계속해서 연결을 유지 → HTTP에 비해 더 많은 리소스 소모
- 클라이언트와 서버가 실시간으로 계속하여 데이터를 주고받아야하는 경우에 유리
- 실시간 동영상 스트리밍이나 온라인 게임 등에 사용

NCHAT Dev - socket()?



△ Fig.1 Flow Diagram for BSD Sockets Communication using TCP

• Server

socket() : 소켓 만들기

bind() : IP주소와 소켓을 묶기

listen() : 클라이언트의 소켓 기다리기

accept() : 클라이언트의 접속 요청 수락하기

send() / recv() : 데이터를 전송하거나 받기

close() : 소켓 닫기

• Client

socket() : 소켓 만들기

connect() : 서버 소켓에 연결 시도하기

send() / recv() : 데이터를 전송하거나 받기

close() : 소켓 닫기

NCHAT Dev - Source Code

```
Server.py > ...
1 import socket
2 import threading
3
4 # 서버 IP/PORT
5 HOST = '127.0.0.1'
6 PORT = 11432
7
8 clients = []
9 lock = threading.Lock()
10
11 # 서버 소켓 생성
12 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 소켓 유형 설정
13 server_socket.bind((HOST, PORT)) # IP 주소와 포트를 소켓으로 묶기
14 server_socket.listen() # 클라이언트의 소켓 기다리기
15
16 print(f"서버가 {HOST}:{PORT}에서 대기 중입니다.")
17
18 # 클라이언트 - 서버 연결 관리
19 def handle_client(client_socket, addr):
20     with lock:
21         clients.append(client_socket)
22
23     while True:
24         data = client_socket.recv(1024).decode() # 데이터 받기
25         if not data:
26             break
27         data = f"[{addr[0]}:{addr[1]}] : " + data
28         print(data)
29         broadcast(data, client_socket)
30
31     with lock:
32         clients.remove(client_socket)
33     client_socket.close() # 소켓 종료
```

```
34
35     # 모든 클라이언트에게 메시지 전송(Broadcast)
36     def broadcast(message, client_socket):
37         with lock:
38             for client in clients:
39                 if client != client_socket:
40                     try:
41                         client.send(message.encode()) # 데이터 보내기
42                     except:
43                         client.close()
44                         clients.remove(client)
45
46
47     while True: # Main 쓰레드
48         client_socket, addr = server_socket.accept()
49         print(f"새로운 연결: {addr[0]}:{addr[1]}")
50         client_handler = threading.Thread(target=handle_client, args=(client_socket, addr))
51         client_handler.start()
52
```

● Server.py

- BroadCast 기능 구현 및 treading 모듈을 활용하여
다수 이용자가 참여 가능하도록 설계
- treading.Lock을 활용하여 데이터 무결성 유지 및
경쟁 상태(Race Condition) 문제 해결

NCHAT Dev - Source Code

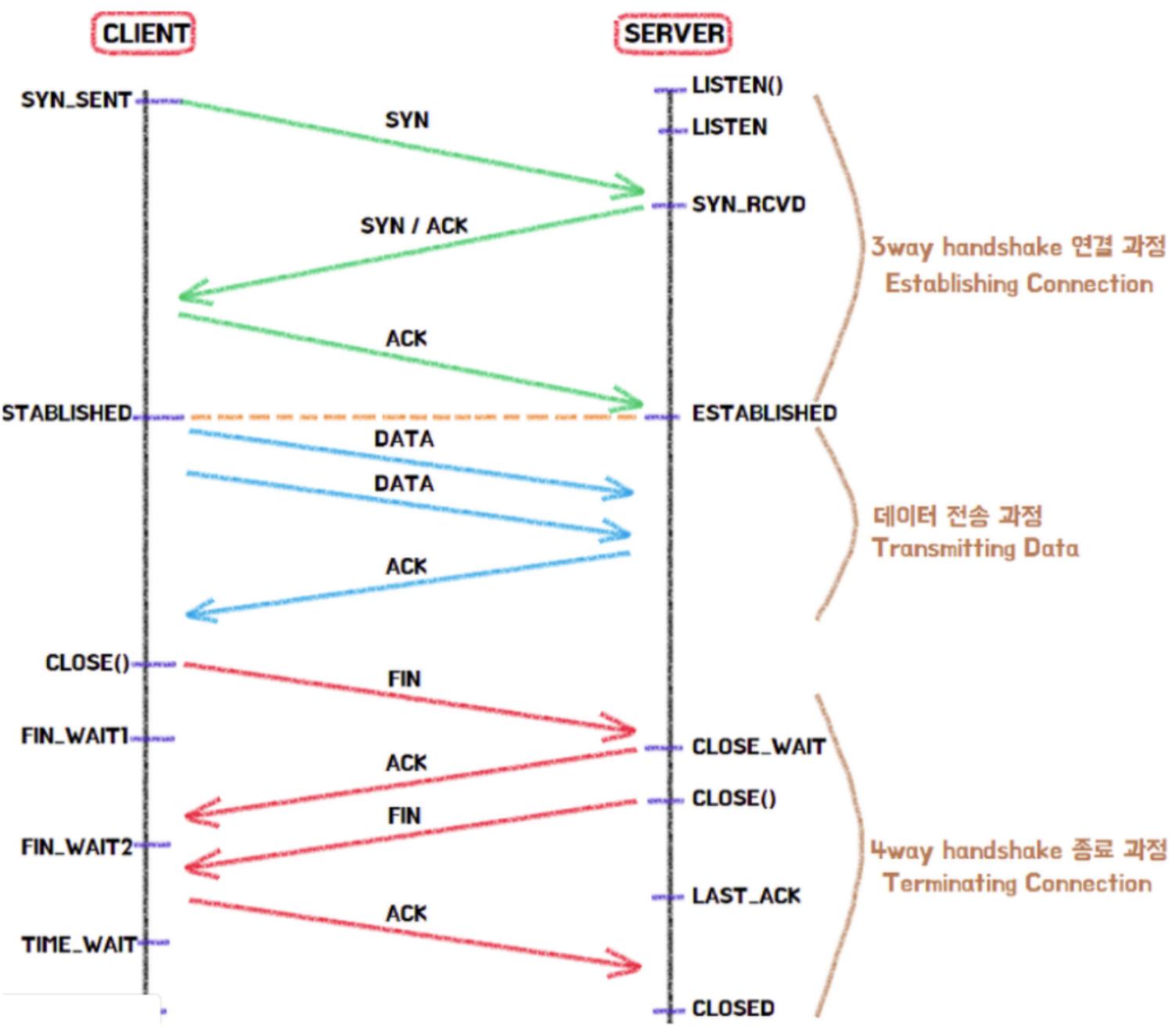
```
Client.py > ...
1 import socket
2 import sys
3 from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QTextEdit, QVBoxLayout, QWidget, QLineEdit
4 import threading
5
6 # 서버 IP/PORT 설정
7 HOST = '127.0.0.1'
8 PORT = 11432
9
10 class ChatClient(QMainWindow):
11     def __init__(self):
12         super().__init__()
13
14         self.initUI()
15         self.initSocket()
16     #PyQt5 UI
17     def initUI(self):
18         self.setWindowTitle('Chat 프로그램')
19         self.setGeometry(100, 100, 400, 400)
20
21         self.central_widget = QWidget(self)
22         self.setCentralWidget(self.central_widget)
23
24         self.layout = QVBoxLayout()
25
26         self.text_box = QTextEdit(self)
27         self.text_box.setReadOnly(True)
28         self.layout.addWidget(self.text_box)
29
30         self.input_box = QLineEdit(self)
31         self.input_box.returnPressed.connect(self.send_message)
32         self.layout.addWidget(self.input_box)
33
34         self.send_button = QPushButton('보내기', self)
35         self.send_button.clicked.connect(self.send_message)
36         self.layout.addWidget(self.send_button)
37
38         self.central_widget.setLayout(self.layout)
```

● Client.py

- PyQt5를 이용하여
NCHAT GUI 구성
- treading 모듈을 활용하여
동시 실행을 통한 UI 프리징 방지

```
39     #클라이언트 소켓 생성
40     def initSocket(self):
41         self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
42         self.client_socket.connect((HOST, PORT)) # 소켓 연결
43
44         self.receive_thread = threading.Thread(target=self.receive_message)
45         self.receive_thread.daemon = True
46         self.receive_thread.start()
47     #메시지 수신
48     def receive_message(self):
49         while True:
50             try:
51                 message = self.client_socket.recv(1024).decode() # 소켓 수신
52                 self.text_box.append(message)
53             except Exception as e:
54                 print(e)
55                 break
56     # 메시지 전송
57     def send_message(self):
58         message = self.input_box.text()
59         if message:
60             self.client_socket.send(message.encode()) # 소켓 송신
61             self.input_box.clear()
62             message = "(ME) : " + message
63             self.text_box.append(message)
64
65     if __name__ == '__main__':
66         app = QApplication(sys.argv)
67         client = ChatClient()
68         client.show()
69         sys.exit(app.exec_())
70
71
```

NCHAT Dev - 3 Way Handshake?



△ Fig.2 Flow Diagram for TCP 3 Way Handshake

● 3 Way Handshake?

클라이언트 - 서버 간 최초 연결 시 필요한 과정.

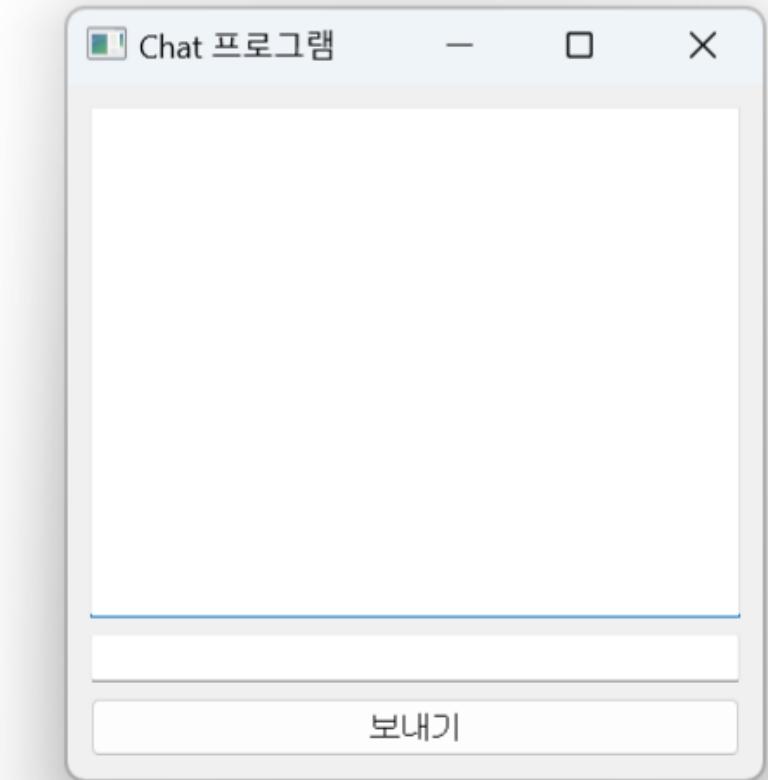
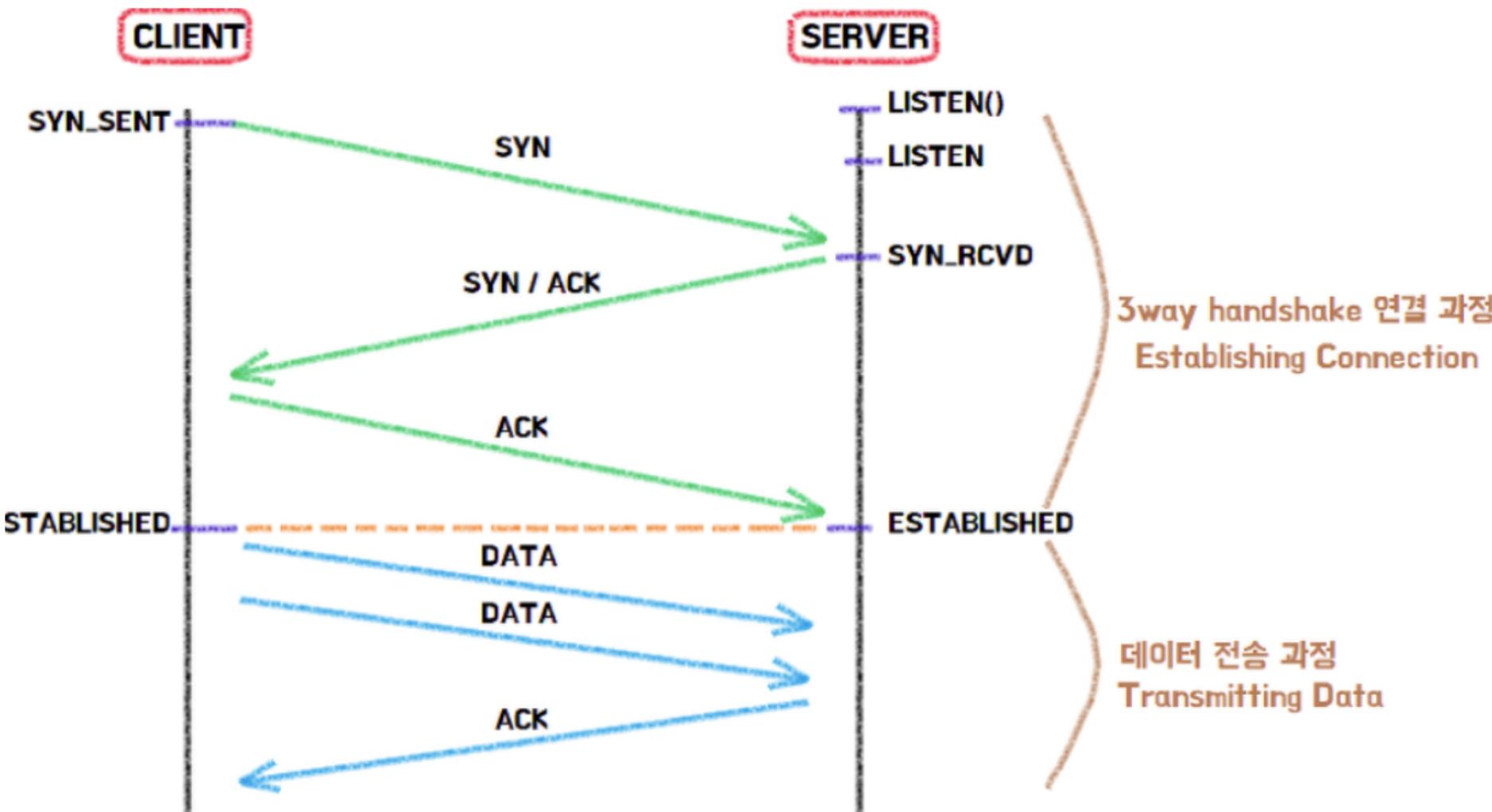
● 과정

1. 클라이언트는 서버로 통신을 시작하겠다는 SYN을 보냄.
2. 서버는 그에 대한 응답으로 SYN+ACK를 보냄.
3. 마지막으로 클라이언트는 서버로부터 받은 패킷에 대한 응답으로 ACK를 보냄.
=> 3-way-handshake를 정상적으로 마친 다음 클라이언트는 서버에 데이터를 요청한다.

Wireshark를 이용한 NCHAT 패킷 분석

No.	Time	Source	Destination	Protocol	Length	Info
455	3.802384	192.168.104.241	132.226.22.163	TCP	66	57321 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
456	3.819341	132.226.22.163	192.168.104.241	TCP	66	19132 → 57321 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
457	3.819724	192.168.104.241	132.226.22.163	TCP	54	57321 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0

● 3 Way Handshake



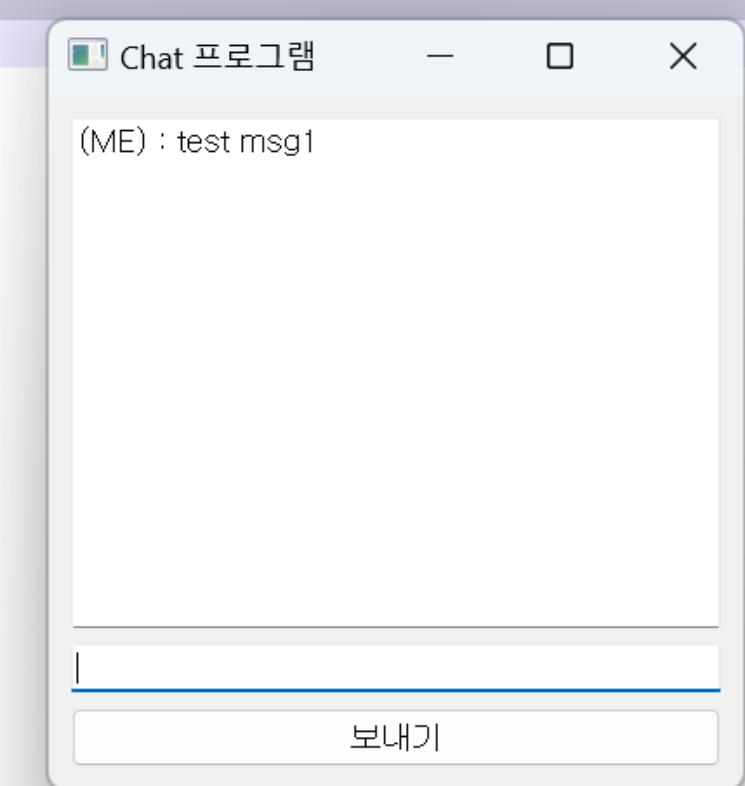
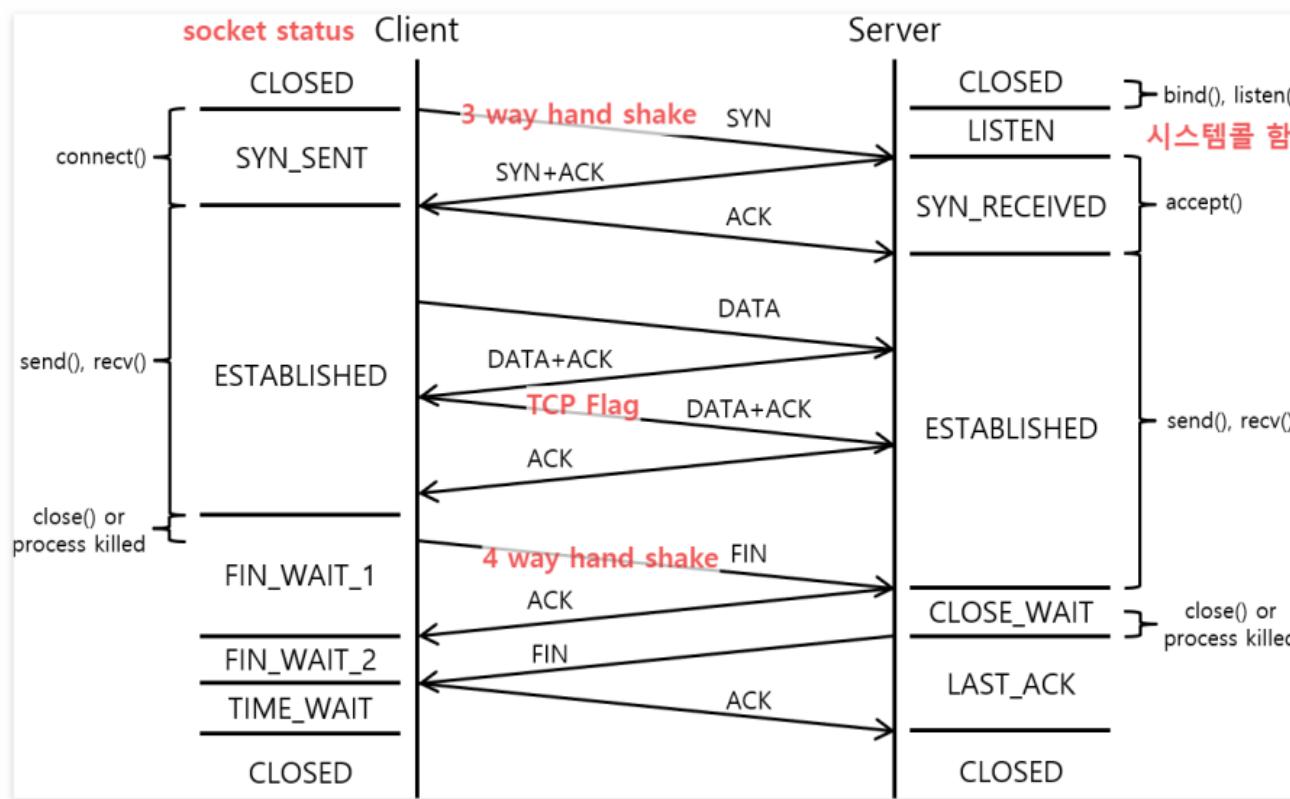
SYN(Synchronization:동기화(동시)) - S : 연결 요청 플래그
- TCP에서 세션을 성립할 때 가장 먼저 보내는 패킷

ACK(Acknowledgement:승인(OK)) - A : 응답 플래그
- 상대방으로부터 패킷을 받았다는 걸 알려주는 패킷,
다른 플래그와 같이 출력되는 경우도 있다.

Wireshark를 이용한 NCHAT 패킷 분석

No.	Time	Source	Destination	Protocol	Length	Info
455	3.802384	192.168.104.241	132.226.22.163	TCP	66	57321 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
456	3.819341	132.226.22.163	192.168.104.241	TCP	66	19132 → 57321 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
457	3.819724	192.168.104.241	132.226.22.163	TCP	54	57321 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
14276	112.091737	192.168.104.241	132.226.22.163	TCP	63	57321 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
14277	112.108224	132.226.22.163	192.168.104.241	TCP	60	19132 → 57321 [ACK] Seq=1 Ack=10 Win=64256 Len=0

- send & receive data
DATA + ACK - ACK



```

> Frame 14276: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface \Device\NPF_{F8F2114A-BF
> Ethernet II, Src: Intel_5d:6d:da (f8:9e:94:5d:6d:da), Dst: Secuicom_c5:ed:55 (00:05:66:c5:ed:55)
> Internet Protocol Version 4, Src: 192.168.104.241, Dst: 132.226.22.163
> Transmission Control Protocol, Src Port: 57321, Dst Port: 19132, Seq: 1, Ack: 1, Len: 9
< Data (9 bytes)
  Data: 74657374206d736731
  [Length: 9]

```

0000	00 05 66 c5 ed 55 f8 9e 94 5d 6d da 08 00 45 00	..f..U.. .]m..E..
0010	00 31 2f be 40 00 80 06 00 00 c0 a8 68 f1 84 e2	.1/@... .h...
0020	16 a3 df e9 4a bc 00 13 41 c3 12 e5 ac 75 50 18J... A...uP..
0030	02 01 10 1e 00 00 74 65 73 74 20 6d 73 67 31te st msg1

Wireshark를 이용한 NCHAT 패킷 분석

ip.addr == 132.226.22.163

No.	Time	Source	Destination	Protocol	Length	Info
5359	44.542737	192.168.104.241	132.226.22.163	TCP	66	49694 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5360	44.555328	132.226.22.163	192.168.104.241	TCP	66	19132 → 49694 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5361	44.555602	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
18678	155.918455	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
18679	155.931920	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=1 Ack=10 Win=64256 Len=0
35380	255.229948	192.168.104.241	132.226.22.163	TCP	66	50903 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
35381	255.246453	132.226.22.163	192.168.104.241	TCP	66	19132 → 50903 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
35382	255.246740	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
39195	282.676562	192.168.104.241	132.226.22.163	TCP	63	50903 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
39196	282.692465	132.226.22.163	192.168.104.241	TCP	60	19132 → 50903 [ACK] Seq=1 Ack=10 Win=64256 Len=0
39197	282.696451	132.226.22.163	192.168.104.241	TCP	87	19132 → 49694 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
39206	282.749834	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=10 Ack=34 Win=131328 Len=0

Chat 프로그램

(ME) : test msg1
(210.95.79.143:50903) : test msg2

보내기

• PSH(Push) - P : 밀어 넣기

- 응답 속도 중요한 프로그램에 사용

=> L7(응용프로그램 계층)으로 바로 전달

Frame 39197: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface \Device\NPF_{F8F2114A

Ethernet II, Src: Secuicom_c5:ed:55 (00:05:66:c5:ed:55), Dst: Intel_5d:6d:da (f8:9e:94:5d:6d:da)

Internet Protocol Version 4, Src: 132.226.22.163, Dst: 192.168.104.241

Transmission Control Protocol, Src Port: 19132, Dst Port: 49694, Seq: 1, Ack: 10, Len: 33

Source Port: 19132
Destination Port: 49694
[Stream index: 31]

[Conversation completeness: Incomplete, DATA (15)]
[TCP Segment Len: 33]

Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 2193964455
[Next Sequence Number: 34 (relative sequence number)]
Acknowledgment Number: 10 (relative ack number)
Acknowledgment number (raw): 1247798760
0101 = Header Length: 20 bytes (5)

Flags: 0x018 (PSH, ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0xe90e [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0

0000 f8 9e 94 5d 6d da 00 05 66 c5 ed 55 08 00 45 00 ...jm... f..U..E.
0010 00 49 42 e0 40 00 32 06 40 b0 84 e2 16 a3 c0 a8 .IB@-2 @.....
0020 68 f1 4a bc c2 1e 82 c5 3d a7 4a 5f e5 e8 50 18 h.J..... =J..P.
0030 01 f6 e9 0e 00 00 28 32 31 30 2e 39 35 2e 37 39(2 10.95.79
0040 2e 31 34 33 3a 35 30 39 30 33 29 20 3a 20 74 65 .143:509 03) : te
0050 73 74 20 6d 73 67 32 st msg2

Chat 프로그램

(ME) : test msg2

보내기

Wireshark를 이용한 NCHAT 패킷 분석

ip.addr == 132.226.22.163

No.	Time	Source	Destination	Protocol	Length	Info
5359	44.542737	192.168.104.241	132.226.22.163	TCP	66	49694 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5360	44.555328	132.226.22.163	192.168.104.241	TCP	66	19132 → 49694 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5361	44.555602	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
18678	155.918455	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
18679	155.931920	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=1 Ack=10 Win=64256 Len=0
35380	255.229948	192.168.104.241	132.226.22.163	TCP	66	50903 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
35381	255.246453	132.226.22.163	192.168.104.241	TCP	66	19132 → 50903 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
35382	255.246740	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
39195	282.676562	192.168.104.241	132.226.22.163	TCP	63	50903 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
39196	282.692465	132.226.22.163	192.168.104.241	TCP	60	19132 → 50903 [ACK] Seq=1 Ack=10 Win=64256 Len=0
39197	282.696451	132.226.22.163	192.168.104.241	TCP	87	19132 → 49694 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
39206	282.749834	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=10 Ack=34 Win=131328 Len=0
82195	519.203234	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=10 Ack=34 Win=131328 Len=9
82208	519.302486	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=34 Ack=19 Win=64256 Len=0
82209	519.302486	132.226.22.163	192.168.104.241	TCP	87	19132 → 50903 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
82210	519.352591	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [ACK] Seq=10 Ack=34 Win=131328 Len=0

Chat 프로그램

```
(ME) : test msg1
(210.95.79.143:50903) : test msg2
(ME) : test msg3
```

보내기

> Frame 82195: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface \Device\NPF_{F8F2114A
 > Ethernet II, Src: Intel_5d:6d:da (f8:9e:94:5d:6d:da), Dst: Secuicom_c5:ed:55 (00:05:66:c5:ed:55)
 > Internet Protocol Version 4, Src: 192.168.104.241, Dst: 132.226.22.163
 > Transmission Control Protocol, Src Port: 49694, Dst Port: 19132, Seq: 10, Ack: 34, Len: 9
 Source Port: 49694
 Destination Port: 19132
 [Stream index: 31]
 > [Conversation completeness: Incomplete, DATA (15)]
 [TCP Segment Len: 9]
 Sequence Number: 10 (relative sequence number)
 Sequence Number (raw): 1247798760
 [Next Sequence Number: 19 (relative sequence number)]
 Acknowledgment Number: 34 (relative ack number)
 Acknowledgment number (raw): 2193964488
 0101 = Header Length: 20 bytes (5)
 > Flags: 0x018 (PSH, ACK)
 Window: 513
 [Calculated window size: 131328]
 [Window size scaling factor: 256]
 Checksum: 0x3c44 [unverified]
 [Checksum Status: Unverified]

0000 00 05 66 c5 ed 55 f8 9e 94 5d 6d da 08 00 45 00 ..f..U.. .]m...E.
 0010 00 31 2f b8 40 00 80 06 00 00 c0 a8 68 f1 84 e2 .1/.@... .h...
 0020 16 a3 c2 1e 4a bc 4a 5f e5 e8 82 c5 3d c8 50 18 ...J.J_.=P.
 0030 02 01 3c 44 00 00 74 65 73 74 20 6d 73 67 33 ..<D..te st msg3

Chat 프로그램

```
(ME) : test msg2
(210.95.79.143:49694) : test msg3
```

보내기

Wireshark를 이용한 NCHAT 패킷 분석

ip.addr == 132.226.22.163

No.	Time	Source	Destination	Protocol	Length	Info
5359	44.542737	192.168.104.241	132.226.22.163	TCP	66	49694 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5360	44.555328	132.226.22.163	192.168.104.241	TCP	66	19132 → 49694 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5361	44.555602	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
18678	155.918455	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
18679	155.931920	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=1 Ack=10 Win=64256 Len=0
35380	255.229948	192.168.104.241	132.226.22.163	TCP	66	50903 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
35381	255.246453	132.226.22.163	192.168.104.241	TCP	66	19132 → 50903 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
35382	255.246740	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
39195	282.676562	192.168.104.241	132.226.22.163	TCP	63	50903 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
39196	282.692465	132.226.22.163	192.168.104.241	TCP	60	19132 → 50903 [ACK] Seq=1 Ack=10 Win=64256 Len=0
39197	282.696451	132.226.22.163	192.168.104.241	TCP	87	19132 → 49694 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
39206	282.749834	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=10 Ack=34 Win=131328 Len=0
82195	519.203234	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=10 Ack=34 Win=131328 Len=9
82208	519.302486	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=34 Ack=19 Win=64256 Len=0
82209	519.302486	132.226.22.163	192.168.104.241	TCP	87	19132 → 50903 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
82210	519.352591	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [ACK] Seq=10 Ack=34 Win=131328 Len=0

Chat 프로그램

```
(ME) : test msg1
(210.95.79.143:50903) : test msg2
(ME) : test msg3
```

보내기

```
> Frame 82209: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface \Device\NPF_{F8F2114A
> Ethernet II, Src: Secuicom_c5:ed:55 (00:05:66:c5:ed:55), Dst: Intel_5d:6d:da (f8:9e:94:5d:6d:da)
> Internet Protocol Version 4, Src: 132.226.22.163, Dst: 192.168.104.241
< Transmission Control Protocol, Src Port: 19132, Dst Port: 50903, Seq: 1, Ack: 10, Len: 33
  Source Port: 19132
  Destination Port: 50903
  [Stream index: 172]
  > [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 33]
    Sequence Number: 1      (relative sequence number)
    Sequence Number (raw): 1323399164
    [Next Sequence Number: 34      (relative sequence number)]
    Acknowledgment Number: 10      (relative ack number)
    Acknowledgment number (raw): 3428296385
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 502
    [Calculated window size: 64256]
    [Window size scaling factor: 128]
    Checksum: 0x8816 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

0000	f8 9e 94 5d 6d da 00 05	66 c5 ed 55 08 00 45 00	...]m... f..U..E..
0010	00 49 f3 96 40 00 32 06	8f f9 84 e2 16 a3 c0 a8	I..@.2.
0020	68 f1 4a bc c6 d7 4e e1	77 fc cc 57 a6 c1 50 18	h..J..N.. w..W..P..
0030	01 f6 88 16 00 00 28 32	31 30 2e 39 35 2e 37 39(2 10.95.79
0040	2e 31 34 33 3a 34 39 36	39 34 29 20 3a 20 74 65	.143:496 94) : te
0050	73 74 20 6d 73 67 33		st msg3

Chat 프로그램

```
(ME) : test msg2
(210.95.79.143:49694) : test msg3
```

보내기

Wireshark를 이용한 NCHAT 패킷 분석

ip.addr == 132.226.22.163						
No.	Time	Source	Destination	Protocol	Length	Info
5359	44.542737	192.168.104.241	132.226.22.163	TCP	66	49694 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5360	44.555328	132.226.22.163	192.168.104.241	TCP	66	19132 → 49694 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5361	44.555602	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
18678	155.918455	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
18679	155.931920	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=1 Ack=10 Win=64256 Len=0
35380	255.229948	192.168.104.241	132.226.22.163	TCP	66	50903 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
35381	255.246453	132.226.22.163	192.168.104.241	TCP	66	19132 → 50903 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
35382	255.246740	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
39195	282.676562	192.168.104.241	132.226.22.163	TCP	63	50903 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
39196	282.692465	132.226.22.163	192.168.104.241	TCP	60	19132 → 50903 [ACK] Seq=1 Ack=10 Win=64256 Len=0
39197	282.696451	132.226.22.163	192.168.104.241	TCP	87	19132 → 49694 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
39206	282.749834	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=10 Ack=34 Win=131328 Len=0
82195	519.203234	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=10 Ack=34 Win=131328 Len=9
82208	519.302486	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=34 Ack=19 Win=64256 Len=0
82209	519.302486	132.226.22.163	192.168.104.241	TCP	87	19132 → 50903 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
1688...	1109.389308	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [RST, ACK] Seq=19 Ack=34 Win=0 Len=0
1689...	1112.991426	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [RST, ACK] Seq=10 Ack=34 Win=0 Len=0

> Frame 168959: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{F8F2114A-E...
 > Ethernet II, Src: Intel_5d:6d:da (f8:9e:94:5d:6d:da), Dst: Secuicom_c5:ed:55 (00:05:66:c5:ed:55)
 > Internet Protocol Version 4, Src: 192.168.104.241, Dst: 132.226.22.163
 > Transmission Control Protocol, Src Port: 50903, Dst Port: 19132, Seq: 10, Ack: 34, Len: 0

Source Port: 50903
 Destination Port: 19132
 [Stream index: 172]
 > [Conversation completeness: Complete, WITH_DATA (47)]
 [TCP Segment Len: 0]
 Sequence Number: 10 (relative sequence number)
 Sequence Number (raw): 3428296385
 [Next Sequence Number: 10 (relative sequence number)]
 Acknowledgment Number: 34 (relative ack number)
 Acknowledgment number (raw): 1323399197
 0101 ... = Header Length: 20 bytes (5)
 > Flags: 0x014 (RST, ACK)

0000 00 05 66 c5 ed 55 f8 9e 94 5d 6d da 08 00 45 00 ..f..U.. .]m...E.
 0010 00 28 2f bb 40 00 80 06 00 00 c0 a8 68 f1 84 e2 .(/@... .h...
 0020 16 a3 c6 d7 4a bc cc 57 a6 c1 4e e1 78 1d 50 14 ..J..W ..N.x.P.
 0030 00 00 9f 05 00 00 ..

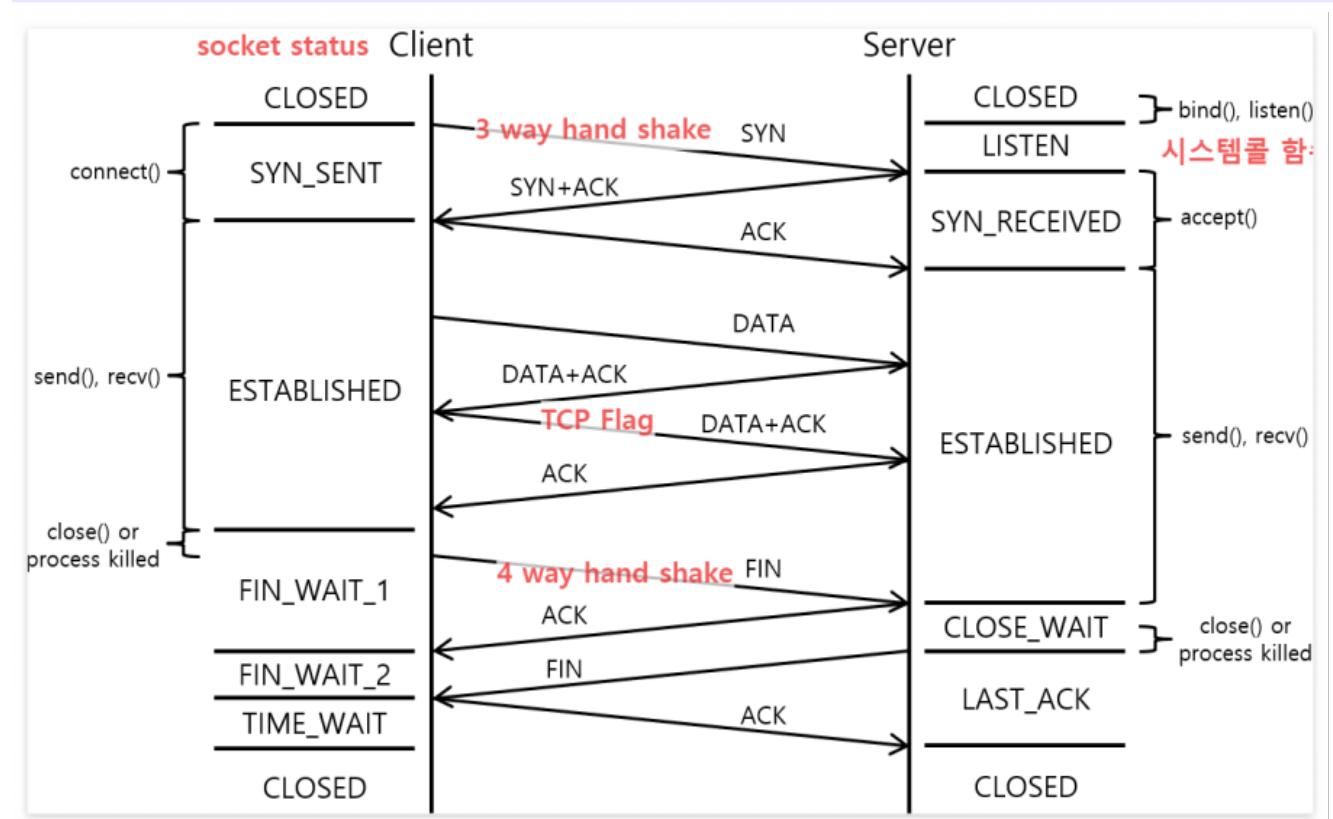
- **RST(Reset) Flag - 재 연결 종료**
 - 재설정 과정
 - 비정상적인 연결 끊기 과정에 해당됨.

보안 취약점 분석 및 프로그램 개선 방안

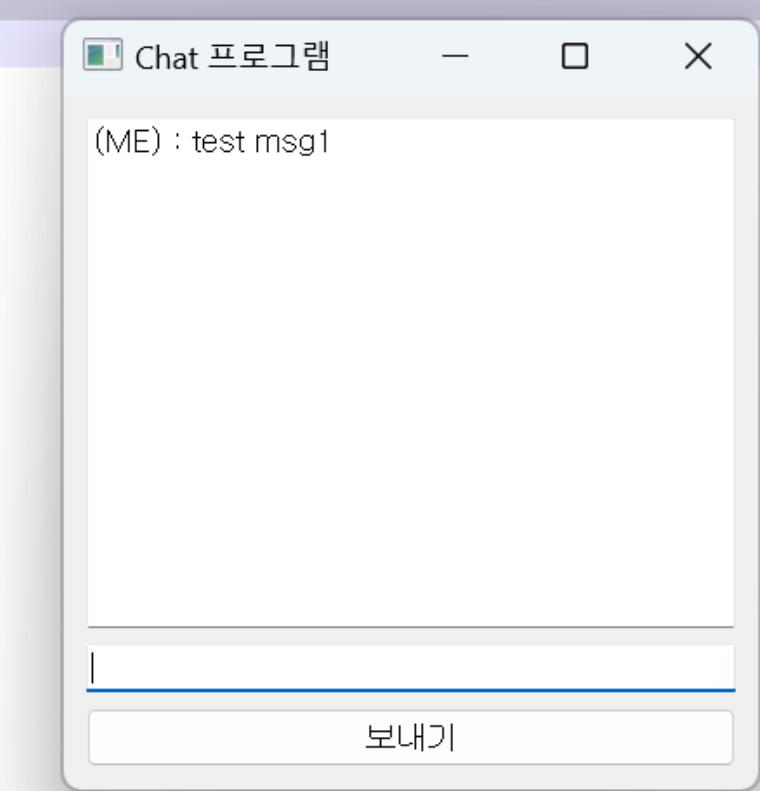


보안 취약점 분석 및 프로그램 개선 방안

No.	Time	Source	Destination	Protocol	Length	Info
455	3.802384	192.168.104.241	132.226.22.163	TCP	66	57321 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
456	3.819341	132.226.22.163	192.168.104.241	TCP	66	19132 → 57321 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
457	3.819724	192.168.104.241	132.226.22.163	TCP	54	57321 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
14276	112.091737	192.168.104.241	132.226.22.163	TCP	63	57321 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
14277	112.108224	132.226.22.163	192.168.104.241	TCP	60	19132 → 57321 [ACK] Seq=1 Ack=10 Win=64256 Len=0



● 통신 암호화 X
=> Wireshark와 같은 프로그램으로
패킷 스니핑 시 데이터 확인/변조 가능



```

> Frame 14276: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface \Device\NPF_{F8F2114A-BF
> Ethernet II, Src: Intel_5d:6d:da (f8:9e:94:5d:6d:da), Dst: Secuicom_c5:ed:55 (00:05:66:c5:ed:55)
> Internet Protocol Version 4, Src: 192.168.104.241, Dst: 132.226.22.163
> Transmission Control Protocol, Src Port: 57321, Dst Port: 19132, Seq: 1, Ack: 1, Len: 9
< Data (9 bytes)
  Data: 74657374206d736731
  [Length: 9]

```

0000	00 05 66 c5 ed 55 f8 9e 94 5d 6d da 08 00 45 00	..f..U.. .]m.. E..
0010	00 31 2f be 40 00 80 06 00 00 c0 a8 68 f1 84 e2	.1/@... .h...
0020	16 a3 df e9 4a bc 00 13 41 c3 12 e5 ac 75 50 18J... A... uP..
0030	02 01 10 1e 00 00 74 65 73 74 20 6d 73 67 31te st msg1

보안 취약점 분석 및 프로그램 개선 방안

The screenshot shows the CyberChef interface. On the left, the sidebar has a blue header 'Cyber Chef' with a chef's hat icon. Below it are 'My Request' and 'Request Bin'. The main area has tabs for 'Operations', 'Favourites' (with a star icon), and 'Data format'. Under 'Operations', there are several options: 'To Base64', 'From Base64', 'To Hex', 'From Hex', 'To Hexdump', 'From Hexdump', 'URL Decode', 'Regular expression', 'Entropy', 'Fork', 'Magic', 'Data format', and 'Encryption / Encoding'. The 'From Hex' tab is currently selected. In the center, under the 'Recipe' section, there is a green box labeled 'From Hex' with a 'Delimiter' dropdown set to 'Auto'. To the right, the 'Input' section shows the hex value '74657374206d736731' in a red-bordered input field. The 'Output' section shows the decoded text 'test msg1' in a red-bordered output field. At the bottom, there is a status bar with 'REC 18' and 'LEN 1'.

- 단순히 HEX 값을 복호화 하는 것으로 데이터를 읽을 수 있다.

보안 취약점 분석 및 프로그램 개선 방안

Solution - Cryptography 모듈을 활용한 RSA 암호화 적용

보안 취약점 분석 및 프로그램 개선 방안

```
44     # 클라이언트 소켓 및 RSA 키 생성
45     def initSocket(self):
46         self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
47         self.client_socket.connect((HOST, PORT))
48
49         # RSA 키 쌍 생성
50         self.private_key = rsa.generate_private_key(
51             public_exponent=65537,
52             key_size=2048,
53             backend=default_backend()
54         )
55
56         # 키 직렬화
57         self.public_key = self.private_key.public_key()
58         serialized_public_key = self.public_key.public_bytes(
59             encoding=serialization.Encoding.PEM,
60             format=serialization.PublicFormat.SubjectPublicKeyInfo
61         )
62
63         # RSA 키 전송
64         self.client_socket.sendall(serialized_public_key)
65
66         # RSA 키 교환 수행
67         def perform_key_exchange(self):
68             # 서버 Public key 수신
69             serialized_server_public_key = self.client_socket.recv(2048)
70             server_public_key = serialization.load_pem_public_key(
71                 serialized_server_public_key,
72                 backend=default_backend()
73             )
74
75             self.server_public_key = server_public_key
76
77             # 메시지 수신
78             def receive_message(self):
79                 while True:
80                     try:
81                         encrypted_data = self.client_socket.recv(2048)
82                         if not encrypted_data:
83                             break
84
85                         decrypted_data = self.decrypt(encrypted_data)
86                         self.text_box.append(decrypted_data)
87                     except Exception as e:
88                         print(e)
89                         break
90
91             # 메시지 전송 및 RSA 암호화
92             def send_message(self):
93                 message = self.input_box.text()
94
95                 if message.lower() == 'quit':
96                     self.client_socket.send(message.encode())
97                     self.client_socket.shutdown(socket.SHUT_WR)
98                     self.client_socket.close()
99                     sys.exit("NCHAT has ended.")
100
101             elif message:
102                 encrypted_message = self.encrypt(message, self.server_public_key)
103                 self.client_socket.sendall(encrypted_message)
104                 self.input_box.clear()
105                 message = "(ME) : " + message
106                 self.text_box.append(message)
107
108             # RSA 암호화
109             def encrypt(self, message, public_key):
110                 encrypted_message = public_key.encrypt(
111                     message.encode(),
112                     padding.OAEP(
113                         mgf=padding.MGF1(algorithm=hashes.SHA256()),
114                         algorithm=hashes.SHA256(),
115                         label=None
116                     )
117                 )
118                 return encrypted_message
119
120             # RSA 복호화
121             def decrypt(self, encrypted_message):
122                 decrypted_message = self.private_key.decrypt(
123                     encrypted_message,
124                     padding.OAEP(
125                         mgf=padding.MGF1(algorithm=hashes.SHA256()),
126                         algorithm=hashes.SHA256(),
127                         label=None
128                     )
129                 )
130                 return decrypted_message.decode()
131
132             if __name__ == '__main__':
133                 app = QApplication(sys.argv)
134                 client = ChatClient()
135                 client.show()
136                 sys.exit(app.exec_())
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1580
1
```

보안 취약점 분석 및 프로그램 개선 방안

```
5  from cryptography.hazmat.primitives import serialization, hashes
6  from cryptography.hazmat.primitives.asymmetric import rsa, padding
7
8  HOST = '0.0.0.0'
9  PORT = 19132
10
11 clients = []
12 lock = threading.Lock()
13
14 # RSA 키 쌍 공유
15 private_key = rsa.generate_private_key(
16     public_exponent=65537,
17     key_size=2048,
18     backend=default_backend()
19 )
20
21 # 키 직렬화
22 public_key = private_key.public_key()
23 serialized_public_key = public_key.public_bytes(
24     encoding=serialization.Encoding.PEM,
25     format=serialization.PublicFormat.SubjectPublicKeyInfo
26 )
27
28 def encrypt(message, client_public_key):
29     # 메시지 암호화
30     encrypted_message = client_public_key.encrypt(
31         message.encode(),
32         padding.OAEP(
33             mgf=padding.MGF1(algorithm=hashes.SHA256()),
34             algorithm=hashes.SHA256(),
35             label=None
36         )
37     )
38     return encrypted_message
39
40 def decrypt(encrypted_message):
41     # 메시지 복호화
42     decrypted_message = private_key.decrypt(
43         encrypted_message,
44         padding.OAEP(
45             mgf=padding.MGF1(algorithm=hashes.SHA256()),
46             algorithm=hashes.SHA256(),
47             label=None
48         )
49     )
50     return decrypted_message.decode()
52
53     with lock:
54         clients.append(client_socket)
55
56     # Public key 공유
57     client_socket.sendall(serialized_public_key)
58
59     # 클라이언트 key 수신
60     serialized_client_public_key = client_socket.recv(2048)
61     client_public_key = serialization.load_pem_public_key(
62         serialized_client_public_key,
63         backend=default_backend()
64     )
65
66     while True:
67         encrypted_data = client_socket.recv(2048)
68         if not encrypted_data:
69             break
70
71         decrypted_data = decrypt(encrypted_data)
72         data = f"({addr[0]}:{addr[1]}) : " + decrypted_data
73         print(data)
74
75         encrypted_data = encrypt(data, client_public_key)
76         broadcast(encrypted_data, client_socket)
77
78     with lock:
79         clients.remove(client_socket)
80         client_socket.close()
81
82     def broadcast(message, client_socket):
83         with lock:
84             for client in clients:
85                 if client != client_socket:
86                     try:
87                         client.sendall(message)
88                     except:
89                         client.close()
90                         clients.remove(client)
91
92     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
93     server_socket.bind((HOST, PORT))
94     server_socket.listen()
95
96     print(f"서버가 {HOST}:{PORT}에서 대기 중입니다.")
97
98     while True:
```

● Server.py

- RSA 키 교환/암복호화 적용
- cryptography 모듈 활용

보안 취약점 분석 및 프로그램 개선 방안

Solution)

No.	Time	Source	Destination	Protocol	Length	Info
3464	18.367812	192.168.102.114	132.226.22.163	TCP	66	60629 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
3469	18.382678	132.226.22.163	192.168.102.114	TCP	66	19132 → 60629 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
3470	18.383096	192.168.102.114	132.226.22.163	TCP	54	60629 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
3476	18.397937	132.226.22.163	192.168.102.114	TCP	505	19132 → 60629 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=451
3492	18.438867	192.168.102.114	132.226.22.163	TCP	54	60629 → 19132 [ACK] Seq=1 Ack=452 Win=130816 Len=0
3583	18.600012	192.168.102.114	132.226.22.163	TCP	505	60629 → 19132 [PSH, ACK] Seq=1 Ack=452 Win=130816 Len=451
3594	18.615134	132.226.22.163	192.168.102.114	TCP	60	19132 → 60629 [ACK] Seq=452 Ack=452 Win=64128 Len=0
5952	27.585442	192.168.102.114	132.226.22.163	TCP	310	60629 → 19132 [PSH, ACK] Seq=452 Ack=452 Win=130816 Len=256
5956	27.604066	132.226.22.163	192.168.102.114	TCP	60	19132 → 60629 [ACK] Seq=452 Ack=708 Win=64128 Len=0

```

> Frame 3583: 505 bytes on wire (4040 bits), 505 bytes captured (4040 bits) on interface \Device\NPF_{F8F2114A
> Ethernet II, Src: Intel_5d:6d:da (f8:9e:94:5d:6d:da), Dst: Secuicom_c5:ed:55 (00:05:66:c5:ed:55)
> Internet Protocol Version 4, Src: 192.168.102.114, Dst: 132.226.22.163
> Transmission Control Protocol, Src Port: 60629, Dst Port: 19132, Seq: 1, Ack: 452, Len: 451
└ Data (451 bytes)
  Data [truncated]: 2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b69473
  [Length: 451]

 0030  01 ff ca 38 00 00 2d 2d 2d 2d 42 45 47 49 4e
 0040  20 50 55 42 4c 49 43 20 4b 45 59 2d 2d 2d 2d
 0050  0a 4d 49 49 42 49 6a 41 4e 42 67 6b 71 68 6b 69
 0060  47 39 77 30 42 41 51 45 46 41 41 4f 43 41 51 38
 0070  41 4d 49 49 42 43 67 4b 43 41 51 45 41 77 48 46
 0080  35 56 72 32 4a 50 41 70 72 42 4e 67 33 66 43 64
 0090  58 0a 59 48 43 76 49 39 67 62 54 45 77 48 45 49
 00a0  2f 32 49 45 45 51 42 50 6e 6d 6d 44 51 30 6a 36
 00b0  31 63 71 6a 59 42 61 41 38 41 4e 6c 77 61 6f 49
 00c0  30 67 58 76 77 33 76 49 4f 6e 2b 68 6d 54 66 43
 00d0  7a 4e 0a 50 53 6d 70 37 59 50 4e 6a 35 72 76 62
 00e0  57 6b 36 57 51 74 6b 6f 6b 69 72 61 61 4b 43 4c
 00f0  6c 76 70 7a 34 32 51 30 5a 64 7a 6d 5a 6a 51 34
 0100  64 59 46 4e 6d 34 37 6a 43 7a 4e 64 67 6f 35 61
 0110  4d 33 70 0a 6c 4e 31 39 64 48 32 4c 6c 66 71 69
 0120  7a 47 4c 36 77 73 4d 6d 6c 75 4f 79 65 31 72 43
 0130  4b 6c 4a 72 5a 6b 6f 6f 30 52 51 46 77 4d 42 75
 0140  74 63 32 4f 2f 6e 4a 48 55 46 6b 45 58 35 6f 50
 0150  4f 59 5a 53 0a 7a 7a 74 33 78 6b 68 65 43 66 45
 0160  79 74 57 49 34 47 52 71 6b 64 66 6e 73 67 41 59
 0170  4d 4b 4f 6c 57 6b 65 46 56 78 62 35 71 7a 48 62
 0180  71 45 4a 41 6c 36 70 70 79 51 6d 54 79 78 78 72
 0190  45 32 71 67 48 0a 71 54 36 52 39 45 4d 45 6e 45
 01a0  7a 52 43 4d 76 64 4a 4f 57 76 6d 47 69 66 57 4e
 01b0  6b 57 73 47 45 57 36 4b 68 37 78 33 62 39 59 33
 01c0  63 43 36 2f 2f 4b 4e 4e 55 75 4b 67 32 66 6e 63
 01d0  44 6f 4e 65 36 77 0a 76 77 49 44 41 51 41 42 0a
 01e0  2d 2d 2d 2d 2d 45 4e 44 20 50 55 42 4c 49 43 20
 01f0  4b 45 59 2d 2d 2d 2d 2d 0a

...8... ---BEGIN
PUBLIC KEY-----
.MIIBIjA NBgkqhki
G9w0BAQE FAOCAQ8
AMIIBCgK CAQEAwHF
5Vr2JPAp rBNg3nCd
X·YHCvI9 gbTEwHEI
/2IEEQBP nmmDQ0j6
1cqjYBaA BANlwaoI
0gXvw3vI On+hmTfc
zN·PSmp7 YPNj5rvb
Wk6WQtko kiraakCL
lvpz42Q0 ZdzmZjQ4
dYFNm47j CzNdgo5a
M3p·1N19 dH2L1fq
zGL6wsMm luOye1rC
K1JrZkoo 0RQFwMBU
tc20/nJH UFkEX5oP
OYZS·zzt 3xkheCfE
ytWI4Grq kdFnsgAY
MKOlWkeF Vxb5qzHb
qEJA16pp yQmTyxxr
E2qgh·qT 6R9EMEnE
zRCMvdJO WvmGifWN
kWsGEW6K h7x3b9Y3
cC6//KNN UuKg2fnc
DoNe6w·v wIDAQAB·
-----END PUBLIC
KEY----- .

```

● RSA Public Key 공유

보안 취약점 분석 및 프로그램 개선 방안

Solution)

No.	Time	Source	Destination	Protocol	Length	Info
3464	18.367812	192.168.102.114	132.226.22.163	TCP	66	60629 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
3469	18.382678	132.226.22.163	192.168.102.114	TCP	66	19132 → 60629 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
3470	18.383096	192.168.102.114	132.226.22.163	TCP	54	60629 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
3476	18.397937	132.226.22.163	192.168.102.114	TCP	505	19132 → 60629 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=451
3492	18.438867	192.168.102.114	132.226.22.163	TCP	54	60629 → 19132 [ACK] Seq=1 Ack=452 Win=130816 Len=0
3583	18.600012	192.168.102.114	132.226.22.163	TCP	505	60629 → 19132 [PSH, ACK] Seq=1 Ack=452 Win=130816 Len=451
3594	18.615134	132.226.22.163	192.168.102.114	TCP	60	19132 → 60629 [ACK] Seq=452 Ack=452 Win=64128 Len=0
5952	27.585442	192.168.102.114	132.226.22.163	TCP	310	60629 → 19132 [PSH, ACK] Seq=452 Ack=452 Win=130816 Len=256
5956	27.604066	132.226.22.163	192.168.102.114	TCP	60	19132 → 60629 [ACK] Seq=452 Ack=708 Win=64128 Len=0

```
> Frame 5952: 310 bytes on wire (2480 bits), 310 bytes captured (2480 bits) on interface \Device\NPF_{F8F2114A
> Ethernet II, Src: Intel_5d:6d:da (f8:9e:94:5d:6d:da), Dst: Secuicom_c5:ed:55 (00:05:66:c5:ed:55)
> Internet Protocol Version 4, Src: 192.168.102.114, Dst: 132.226.22.163
> Transmission Control Protocol, Src Port: 60629, Dst Port: 19132, Seq: 452, Ack: 452, Len: 256
✓ Data (256 bytes)
  Data [truncated]: 2207481de41124d6fdf33e302565a815a97c097921d1a1e881677ea147bdd119eaf133622daa9fb8753ed06
  [Length: 256]
  0000  00 05 66 c5 ed 55 f8 9e  94 5d 6d da 08 00 45 00  ..f..U.. .]m...E.
  0010  01 28 39 d4 40 00 80 06  00 00 c0 a8 66 72 84 e2  .(9 @... ....fr...
  0020  16 a3 ec d5 4a bc 41 d6  3d ff 61 52 b0 d4 50 18  ...J..A.. =aR..P..
  0030  01 ff 04 44 00 00 22 07  48 1d e4 11 24 d6 fd f3  ...D..". H...$...
  0040  3e 30 25 65 a8 15 a9 7c  09 79 21 d1 a1 e8 81 67  >0%e... | .y!...g
  0050  7e a1 47 bd d1 19 ea f1  33 62 2d aa 9f b8 75 3e  ~.G..... 3b....u>
  0060  d0 6d da 74 69 57 b1 5e  35 33 04 19 b9 58 6b 77  .m.tiW.^ 53...Xkw
  0070  33 7b 65 fb ce 49 a8 9d  48 f9 4c 1d 0f 09 8c e3  3{e..I.. H..L.....
  0080  05 d3 06 90 8f 74 b4 7a  f4 29 fc d2 ce e0 8a 85  ....t.z .).....
  0090  07 dc 4c 93 c8 bb 73 31  dc 1e a3 e3 e7 d7 42 54  ..L...s1 .....BT
  00a0  e7 7b 3f 2a 59 07 51 4a  cd f3 28 50 55 3a 32 dc  .{*Y..QJ ..(PU:2.
  00b0  4a 3c 65 2a 2d 7e 18 0a  f0 cb 6f 25 72 b8 26 4c  J<e*~... ..o%r.&L
  00c0  71 18 d5 2c 32 d7 37 7c  33 a4 ed 92 b3 26 3f 2c  q..,2.7| 3....&?
  00d0  ec 67 3f 8f 87 d8 f0 ca  bf be a6 2e 80 a0 77 88  .g?.... ....w.
  00e0  2b 4f d0 48 b8 b0 59 49  f5 2c 5b a6 03 69 ac a3  +0.H..YI .,[...i..
  00f0  f1 74 e5 1c 85 fc dc 7c  23 c4 81 85 9d 2d a4 71  .t..... | #.....
  0100  ac dd 66 70 bb 8d 1d 40  bc cf 74 fa 9b 2f 72 59  ..fp...@ ..t..../rY
  0110  04 b9 73 ee 6c e2 7b 61  b5 28 3d b1 22 44 5d 4b  ..s.l.{a .(="D]K
  0120  cb a3 5b 09 47 3b 01 85  4d a5 01 de 9d b2 58 fb  ..[.G... M....X.
  0130  ec 42 5d 8a 44 32  .B].D2
```

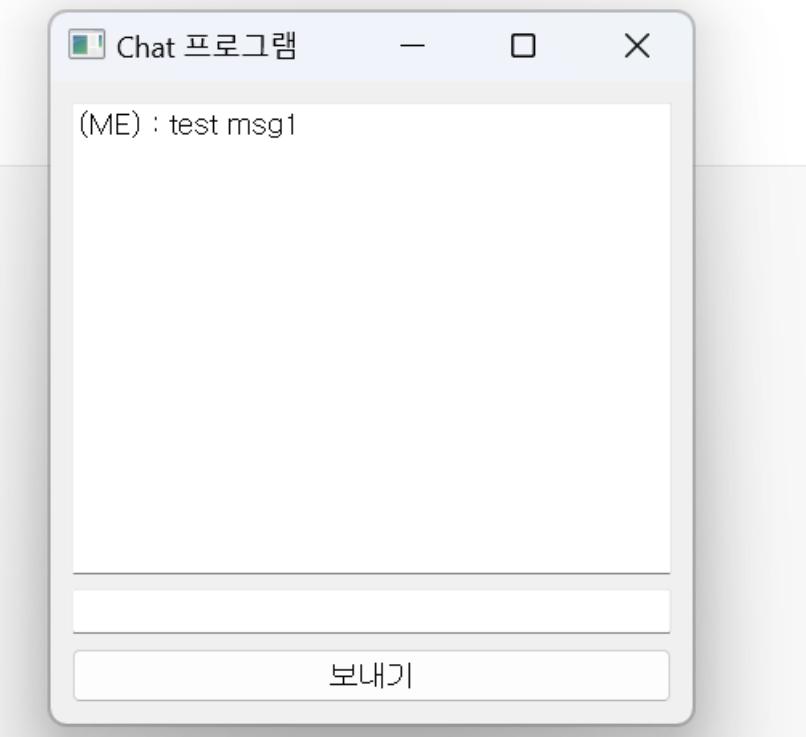
- RSA 암호화되어 스니핑 할 수 없는 데이터

보안 취약점 분석 및 프로그램 개선 방안

Solution)

```
server_encrypt.py > ...
28 def encrypt(message, client_public_key):
29     # 메시지 암호화
30     encrypted_message = client_public_key.encrypt(
31         message.encode(),
32         padding.OAEP(
33             mgf=padding.MGF1(algorithm=hashes.SHA256()),
34             algorithm=hashes.SHA256(),
35             label=None
36         )
37     )
38     return encrypted_message
39
40 def decrypt(encrypted_message):
41     # 메시지 복호화
42     decrypted_message = private_key.decrypt(
43         encrypted_message,
44         padding.OAEP(
45             mgf=padding.MGF1(algorithm=hashes.SHA256()),
46             algorithm=hashes.SHA256(),
47             label=None
48         )
49     )
문제    출력    디버그 콘솔    터미널    포트  21
○ abc@579830b5a52d:~/workspace$ /bin/python3 /config/workspace/server_encrypt.py
서버가 0.0.0.0:19132에서 대기 중입니다.
새로운 연결: 210.95.79.143:60629
(210.95.79.143:60629) : test msg1
```

- 성공적으로 데이터 전송이 이루어진 모습



보안 취약점 분석 및 프로그램 개선 방안

ip.addr == 132.226.22.163						
No.	Time	Source	Destination	Protocol	Length	Info
5359	44.542737	192.168.104.241	132.226.22.163	TCP	66	49694 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5360	44.555328	132.226.22.163	192.168.104.241	TCP	66	19132 → 49694 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5361	44.555602	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
18678	155.918455	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
18679	155.931920	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=1 Ack=10 Win=64256 Len=0
35380	255.229948	192.168.104.241	132.226.22.163	TCP	66	50903 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
35381	255.246453	132.226.22.163	192.168.104.241	TCP	66	19132 → 50903 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
35382	255.246740	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [ACK] Seq=1 Ack=1 Win=131328 Len=0
39195	282.676562	192.168.104.241	132.226.22.163	TCP	63	50903 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=9
39196	282.692465	132.226.22.163	192.168.104.241	TCP	60	19132 → 50903 [ACK] Seq=1 Ack=10 Win=64256 Len=0
39197	282.696451	132.226.22.163	192.168.104.241	TCP	87	19132 → 49694 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
39206	282.749834	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [ACK] Seq=10 Ack=34 Win=131328 Len=0
82195	519.203234	192.168.104.241	132.226.22.163	TCP	63	49694 → 19132 [PSH, ACK] Seq=10 Ack=34 Win=131328 Len=9
82208	519.302486	132.226.22.163	192.168.104.241	TCP	60	19132 → 49694 [ACK] Seq=34 Ack=19 Win=64256 Len=0
82209	519.302486	132.226.22.163	192.168.104.241	TCP	87	19132 → 50903 [PSH, ACK] Seq=1 Ack=10 Win=64256 Len=33
1688...	1109.389308	192.168.104.241	132.226.22.163	TCP	54	49694 → 19132 [RST, ACK] Seq=19 Ack=34 Win=0 Len=0
1689...	1112.991426	192.168.104.241	132.226.22.163	TCP	54	50903 → 19132 [RST, ACK] Seq=10 Ack=34 Win=0 Len=0

> Frame 168959: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{F8F2114A-E...
 > Ethernet II, Src: Intel_5d:6d:da (f8:9e:94:5d:6d:da), Dst: Secuicom_c5:ed:55 (00:05:66:c5:ed:55)
 > Internet Protocol Version 4, Src: 192.168.104.241, Dst: 132.226.22.163
 > Transmission Control Protocol, Src Port: 50903, Dst Port: 19132, Seq: 10, Ack: 34, Len: 0

```

Source Port: 50903
Destination Port: 19132
[Stream index: 172]
> [Conversation completeness: Complete, WITH_DATA (47)]
[TCP Segment Len: 0]
Sequence Number: 10      (relative sequence number)
Sequence Number (raw): 3428296385
[Next Sequence Number: 10      (relative sequence number)]
Acknowledgment Number: 34      (relative ack number)
Acknowledgment number (raw): 1323399197
0101 .... = Header Length: 20 bytes (5)
> Flags: 0x014 (RST, ACK)
...
  
```

```

0000  00 05 66 c5 ed 55 f8 9e  94 5d 6d da 08 00 45 00  ..f..U..  .]m...E..
0010  00 28 2f bb 40 00 80 06  00 00 c0 a8 68 f1 84 e2  .(/ @...  .h...
0020  16 a3 c6 d7 4a bc cc 57  a6 c1 4e e1 78 1d 50 14  ...J..W  .N.x.P..
0030  00 00 9f 05 00 00
  
```

- **RST(Reset) Flag - 재 연결 종료**
 - 재설정 과정
 - 비정상적인 연결 끊기 과정에 해당됨.

보안 취약점 분석 및 프로그램 개선 방안

< RST(Reset) 패킷으로 종료하였을 시, 발생할 수 있는 문제점 >

1. 데이터 손실 발생

=> RST 패킷은 전송된 데이터의 승인 없이 연결을 바로 종료한다.

따라서, 전송 중인 데이터가 전송되지 않고 손실될 수 있다.

2. 리소스 누수 발생

=> 적절한 정리 없이 RST가 사용되는 경우, 메모리와 같은 리소스가 적절하게 해제되지 않아 리소스 누수가 발생할 수 있다.

3. 잠재적인 보안 위험

=> 스퓌핑, 스누핑(스니핑) 공격을 통해 RST 패킷이 악의적인 목적으로 활용되어 잠재적인 보안 위험을 초래할 수 있다.

4. 반쯤 닫힌 연결 상태 지속

=> 한쪽에서 RST 패킷을 전송하면, 상대의 상태와 상관없이 연결을 강제로 종료한다.

따라서, 프로그램에서 예기치 않은 동작이 발생할 수 있다.

==> FIN 패킷을 사용하여, 정상적인 4 Way Handshake 종료 과정을 거쳐야 함.

보안 취약점 분석 및 프로그램 개선 방안

Client.py

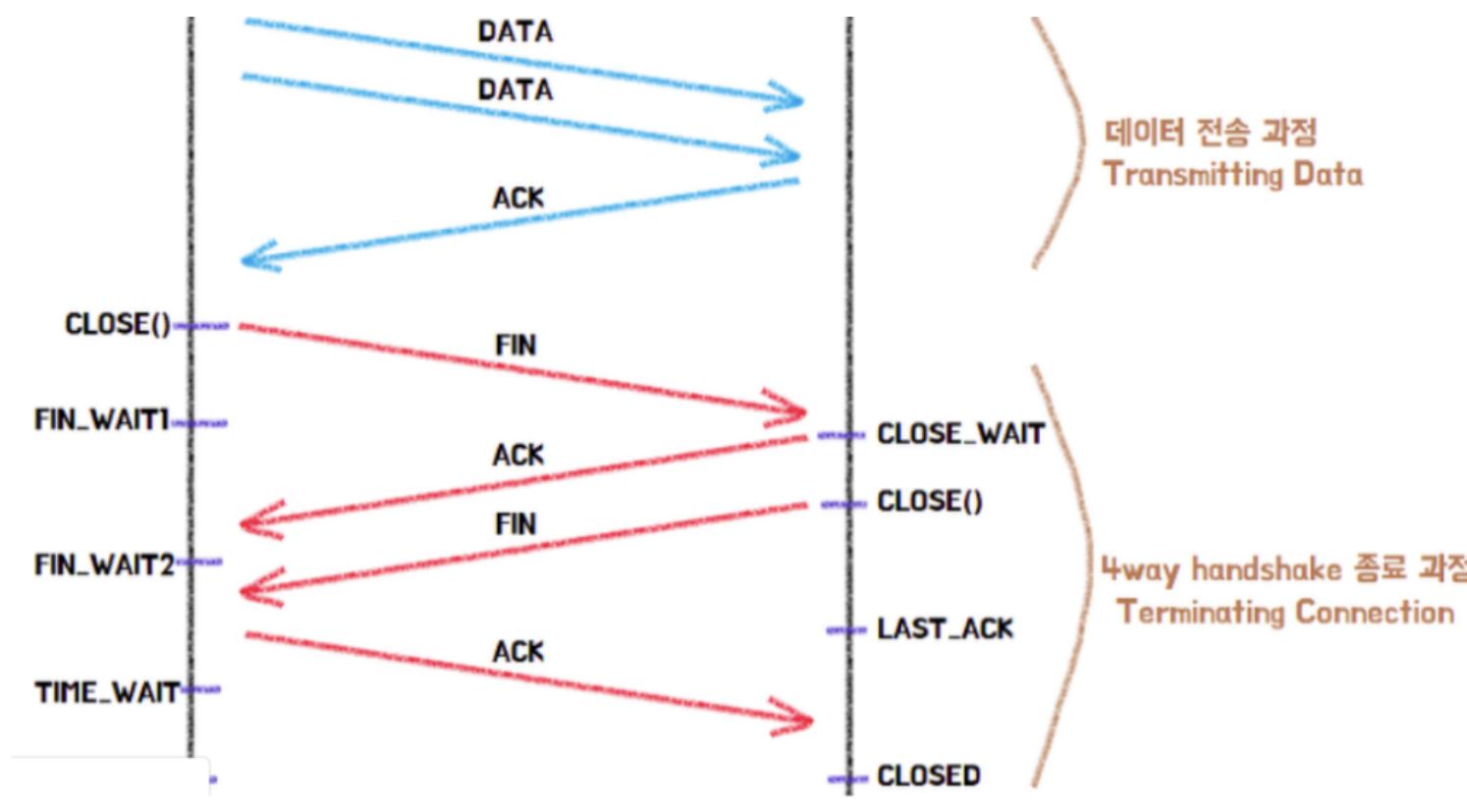
```
56     # 메시지 전송
57     def send_message(self):
58         message = self.input_box.text()
59
59     if message:
60
60         self.client_socket.send(message.encode()) # 소켓 송신
61         self.input_box.clear()
62         message = "(ME) : " + message
63         self.text_box.append(message)
64
```

```
56     # 메시지 전송
57     def send_message(self):
58         message = self.input_box.text()
59
59     if message.lower() == 'quit':
60         self.client_socket.send(message.encode())
61         self.client_socket.shutdown(socket.SHUT_WR) # Send TCP FIN Socket
62         self.client_socket.close()
63         sys.exit("NCHAT has ended.")
64
64     elif message:
65
65         self.client_socket.send(message.encode()) # 소켓 송신
66         self.input_box.clear()
67         message = "(ME) : " + message
68         self.text_box.append(message)
69
```

==> FIN 패킷을 사용하여, 정상적인 4 Way Handshake 종료 과정을 거치도록 코드 수정

보안 취약점 분석 및 프로그램 개선 방안

No.	Time	Source	Destination	Protocol	Length	Info
14	2.876105	172.20.10.2	132.226.22.163	TCP	66	63151 → 19132 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
17	3.360492	132.226.22.163	172.20.10.2	TCP	66	19132 → 63151 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1410 SACK_PERM WS=128
18	3.361189	172.20.10.2	132.226.22.163	TCP	54	63151 → 19132 [ACK] Seq=1 Ack=1 Win=131072 Len=0
47	5.775981	172.20.10.2	132.226.22.163	TCP	59	63151 → 19132 [PSH, ACK] Seq=1 Ack=1 Win=131072 Len=5
50	6.227172	132.226.22.163	172.20.10.2	TCP	54	19132 → 63151 [ACK] Seq=1 Ack=6 Win=64256 Len=0
68	6.841497	172.20.10.2	132.226.22.163	TCP	58	63151 → 19132 [PSH, ACK] Seq=6 Ack=1 Win=131072 Len=4
69	6.841772	172.20.10.2	132.226.22.163	TCP	54	63151 → 19132 [FIN, ACK] Seq=10 Ack=1 Win=131072 Len=0
70	6.891371	132.226.22.163	172.20.10.2	TCP	66	[TCP Dup ACK 50#1] 19132 → 63151 [ACK] Seq=1 Ack=6 Win=64256 Len=0 SLE=10 SRE=11
71	6.891371	132.226.22.163	172.20.10.2	TCP	54	19132 → 63151 [ACK] Seq=1 Ack=11 Win=64256 Len=0
72	6.891371	132.226.22.163	172.20.10.2	TCP	54	19132 → 63151 [FIN, ACK] Seq=1 Ack=11 Win=64256 Len=0
73	6.891598	172.20.10.2	132.226.22.163	TCP	54	63151 → 19132 [ACK] Seq=11 Ack=2 Win=131072 Len=0



==> 정상적인 4 Way Handshake

종료 과정을 거치는 것을 확인할 수 있다.

보안 취약점 분석 및 프로그램 개선 방안

```
Server.py > ...
1 import socket
2 import threading
3
4 # 서버 IP/PORT
5 HOST = '127.0.0.1'
6 PORT = 11432
7
8 clients = []
9 lock = threading.Lock()
10
11 # 서버 소켓 생성
12 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 소켓 유형 설정
13 server_socket.bind((HOST, PORT)) # IP 주소와 포트를 소켓으로 묶기
14 server_socket.listen() # 클라이언트의 소켓 기다리기
15
16 print(f"서버가 {HOST}:{PORT}에서 대기 중입니다.")
17
18 # 클라이언트 - 서버 연결 관리
19 def handle_client(client_socket, addr):
20     with lock:
21         clients.append(client_socket)
22
23     while True:
24         data = client_socket.recv(1024).decode() # 데이터 받기
25         if not data:
26             break
27
28         data = f"[{addr[0]}:{addr[1]}] : " + data
29         print(data)
30         broadcast(data, client_socket)
31
32     with lock:
33         clients.remove(client_socket)
34         client_socket.close() # 소켓 종료
```

```
34
35     # 모든 클라이언트에게 메시지 전송(Broadcast)
36     def broadcast(message, client_socket):
37         with lock:
38             for client in clients:
39                 if client != client_socket:
40                     try:
41                         client.send(message.encode()) # 데이터 보내기
42                     except:
43                         client.close()
44                         clients.remove(client)
45
46
47     while True: # Main 쓰레드
48         client_socket, addr = server_socket.accept()
49         print(f"새로운 연결: {addr[0]}:{addr[1]}")
50         client_handler = threading.Thread(target=handle_client, args=(client_socket, addr))
51         client_handler.start()
52
```

● Server.py

- BroadCast 기능 구현 및 treading 모듈을 활용하여
다수 이용자가 참여 가능하도록 설계
- treading.Lock을 활용하여 데이터 무결성 유지 및
경쟁 상태(Race Condition) 문제 해결

Buffer Overflow

Server.py

```
22
23     while True:
24         data = client_socket.recv(1024).decode() # 데이터 받기
25         if not data:
26             break
27         data = f"({addr[0]}:{addr[1]}) : " + data
28         print(data)
29         broadcast(data, client_socket)
30
31     with lock:
32         clients.remove(client_socket)
33         client_socket.close() # 소켓 종료
```

- 1024 byte 단위로 패킷을 수신함을 알 수 있음(버퍼 크기가 1024 byte)

But, 1024 byte를 넘는 패킷이 서버에 수신된다면?

=> Buffer Overflow가 발생할 수 있다.

Buffer Overflow

Buffer Overflow

메모리를 다루는 데에 오류가 발생하여 잘못된 동작을 하는 것.

버퍼가 훌러넘친다는 말 그대로 버퍼의 처리 과정에서 오류가 발생하여,
데이터가 버퍼 크기를 벗어나게 되고, 벗어난 데이터는 인접 메모리를 덮어 쓰게 되는 현상을 의미함.

Buffer Overflow Attack

프로그램에 버퍼를 조작할 수 있는 버그가 존재할 경우 컴퓨터에게 해커가 원하는 일을 하도록 지시시켜 공격하는 것.

Buffer Overflow

Client.py

```
# 메시지 전송
def send_message(self):
    message = self.input_box.text()
    if message.lower() == 'quit':
        self.client_socket.send(message.encode())
        self.client_socket.shutdown(socket.SHUT_WR) # Send TCP FIN Socket
        self.client_socket.close()
        sys.exit("NCHAT has ended.")
    elif message:
        self.client_socket.send(message.encode()) # 소켓 송신
        self.input_box.clear()
        message = "(ME) : " + message
        self.text_box.append(message)
56     # 메시지 전송
57     def send_message(self):
58         message = self.input_box.text()
59         if message.lower() == 'quit':
60             self.client_socket.send(message.encode())
61             self.client_socket.shutdown(socket.SHUT_WR) # Send TCP FIN Socket
62             self.client_socket.close()
63             sys.exit("NCHAT has ended.")
64         elif message:
65             message = "A" * 4000
66             self.client_socket.send(message.encode()) # 소켓 송신
67             self.input_box.clear()
68             message = "(ME) : " + message
69             self.text_box.append(message)
70
```

<Buffer Overflow Attack>

- UTF-8 기준 영어 1글자당 1byte, ∴ 4000 byte 전송

Buffer Overflow

- 메모리가 침범되어,
데이터가 나누어서 출력됨.

=> 예기치 않은 동작 유발 가능

- C언어의 strcpy, scanf, gets와 같은 함수들은 데이터 영역을 넘어 다른 변수나 함수가 저장된 메모리 영역을 덮어씌울 수 있다.

=> 이를 악용하여 공격에 활용할 수 있어 주의가 필요

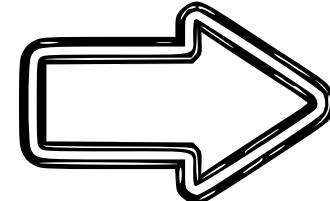
Buffer Overflow

Solution)

```
# 클라이언트 - 서버 연결 관리
def handle_client(client_socket, addr):
    with lock:
        clients.append(client_socket)

    while True:
        data = client_socket.recv(1024).decode() # 데이터 받기
        if not data:
            break
        data = f"({addr[0]}:{addr[1]}) : " + data
        print(data)
        broadcast(data, client_socket)

    with lock:
        clients.remove(client_socket)
    client_socket.close() # 소켓 종료
```



```
# 클라이언트 - 서버 연결 관리
def handle_client(client_socket, addr):
    with lock:
        clients.append(client_socket)

    while True:
        data = b""
        while True:
            chunk = client_socket.recv(1024)
            if not chunk:
                break
            data += chunk
            if len(chunk) < 1024:
                break

        if not data:
            break

        data = f"({addr[0]}:{addr[1]}) : " + data.decode()
        print(data)
        broadcast(data, client_socket)

    with lock:
        clients.remove(client_socket)
    client_socket.close() # 소켓 종료
```

- 수신 패킷 길이를 유동적으로 설정

Buffer Overflow

Solution)

정확하게 'A' 4000개가 출력됨

결론 및 제언

결론 - 1

데이터 입출력(I/O) 부분에서는 보안 취약점이 발생하기 쉽다.
∴ 입출력 부분을 중심으로 코드 점검 필요.

결론 - 2

다양한 보안 취약점은 예기치 않은 오류를 야기할 수 있다. 따라서, 시큐어 코딩을 고려하며 코딩을 할 필요가 있다.

결론 - 3

컴퓨터 구조에 대한 이해 후, 코딩을 할 시 더욱 쉽게 코드를 활용할 수 있다.

제언

완성도 높은 프로그램을 만들기 위하여, 코딩을 할 때 피드백 과정 (QA, 프로그램 테스트, 보안 점검) 을 거칠 필요가 있다고 생각한다.