**NCTU CN2018**

**Lab. 1 – Packet Manipulation via Scapy**

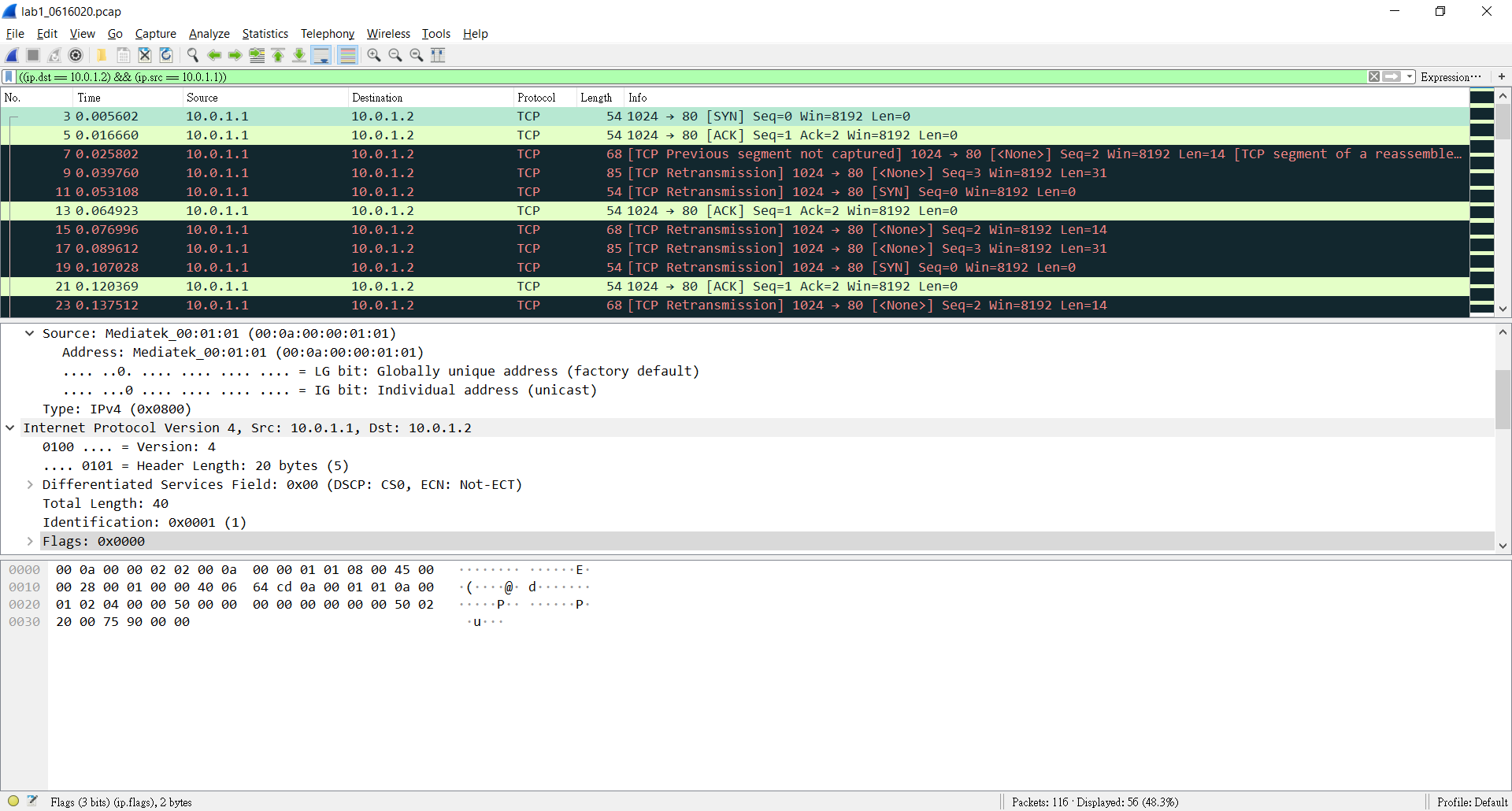
Student name: 鄧佰理Student ID: 0616020 Department: CS

**Part A. Questions**

1. What is your command to filter the packet with customized header on Wireshark?

((ip.dst == 10.0.1.2) && (ip.src == 10.0.1.1))

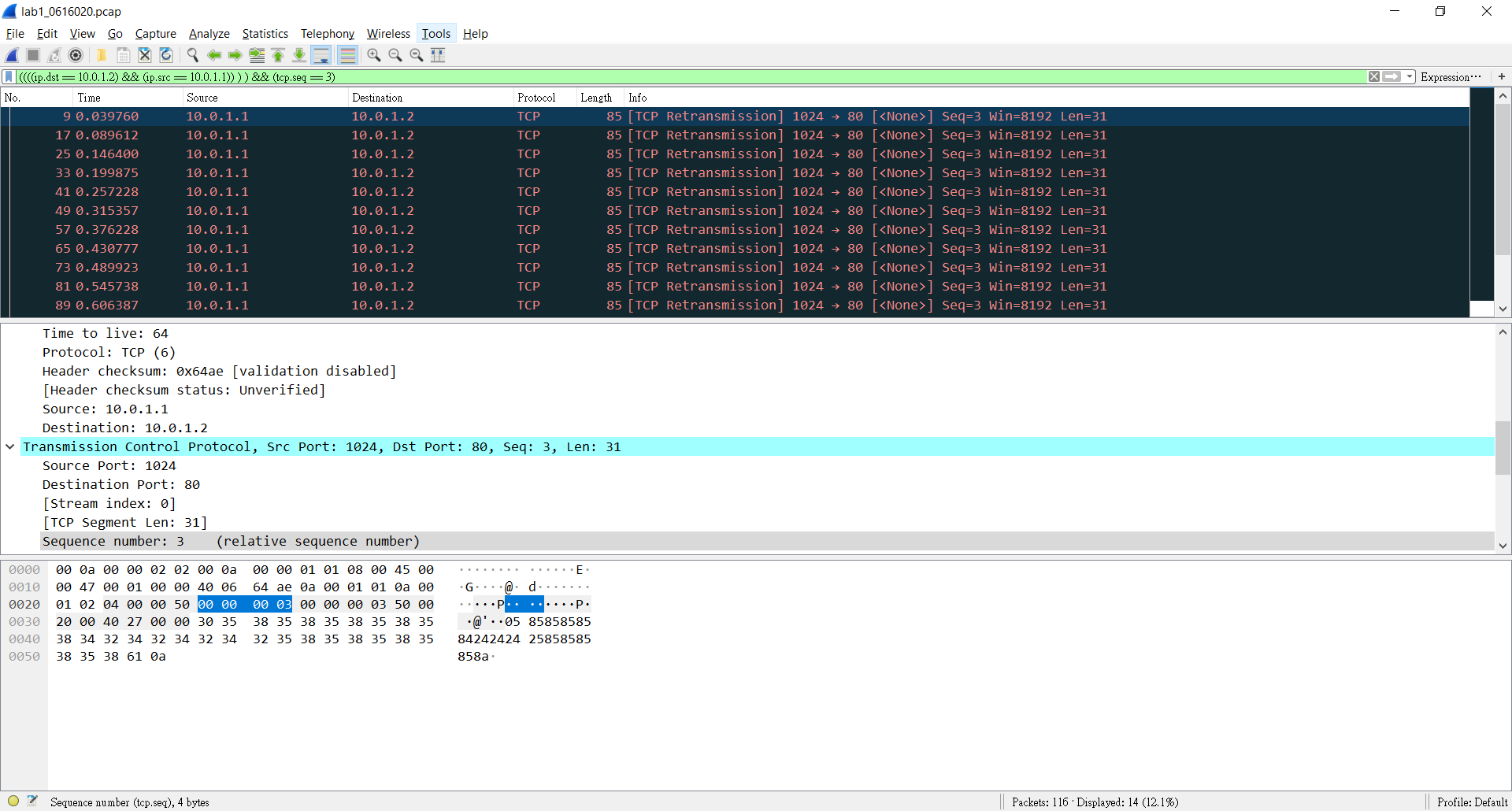
1. Show the screenshot of filtering the packet with customized header.



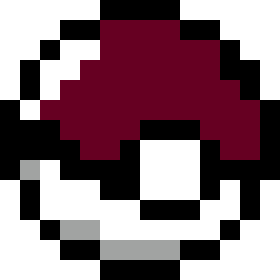
1. What is your command to filter the packet with “secret” payload on Wireshark?

((((ip.dst == 10.0.1.2) && (ip.src == 10.0.1.1)) ) ) && (tcp.seq == 3)

1. Show the screenshot of filtering the packet with “secret” payload.



1. Show the result after decoding the “secret” payload.



Part B. Description

Task 1: Environment Setup

1.Download required files from GitHub and set up Git.

git clone https://github.com/yungshenglu/Packet\_Manipulation

git config --global user.name "Baili Deng"

git config --global user.email "rayman0411@gmail.com"

git remote set-url origin https://github.com/nctucn/lab1-baili0411.git

git push origin master

2.Set up dockerfile by adding the lines of code from the slide.

# Download base image from yungshenglu/ubuntu-env:16.04 (Task 1.)

FROM yungshenglu/ubuntu-env:16.04

# Update software respository (Task 1.)

RUN apt-get update

# Install software repository (Task 1.)

RUN apt-get install –y tcpdump

# Install pip packages (Task 1.)

RUN pip install scapy

# Set the container listens on the specified ports at runtime (Task 1.)

EXPOSE 22

# Clone the repository from GitHub (Task 1.)

RUN git clone https://github.com/yungshenglu/Packet\_Manipulation.git

3.Build image from dockerfile, and build container from image.

If we use Windows, we have to use the console from Kitematic for docker.

# Build the image from Dockerfile

docker build -t cn2018 .

# Build a container named cn2018\_c from cn2018

docker run -d -p 9487:22 --privileged --name cn2018\_c cn2018

# List port 22 mapping on cn2018\_c

docker port cn2018\_c 22

4.If failed, delete container then try again.

# Stop the container that has already named cn2018\_c

docker container stop <container\_id>

# Remove the container named cn2018\_c

docker container rm <container\_id>

5.Connect to container using SSH, SSH works on Windows if using Powershell.

#Check ip\_address of container in Kitematic.

ssh root@<ip\_address> –p 9487

Password: cn2018

6.Add lines of code in ./src/scripts/main.sh to make the script add the h2 (receiver) namespace.

(May have to git clone <https://github.com/yungshenglu/Packet_Manipulation> first if files are not there in container)

# Create h2 network namespaces (Task 1.)

ip netns add h2

# Delete h2 network namespaces (Task 1.)

ip netns del h2

# Bring up the lookup interface in h2 (Task 1.)

ip netns exec h2 ip link set lo up

# Set the interface of h2 to h2-eth0 (Task 1.)

ip link set h2-eth0 netns h2

# Delete the interface of h2-eth0 (Task 1.)

ip link delete h2-eth0

# Activate h2-eth0 and assign IP address (Task 1.)

ip netns exec h2 ip link set dev h2-eth0 up

ip netns exec h2 ip link set h2-eth0 address 00:0a:00:00:02:02

ip netns exec h2 ip addr add 10.0.1.2/24 dev h2-eth0

# Disable all IPv6 on h2-eth0 (Task 1.)

ip netns exec h2 sysctl net.ipv6.conf.h2-eth0.disable\_ipv6=1

# Set the gateway of h2 to 10.0.1.254 (Task 1.)

ip netns exec h2 ip route add default via 10.0.1.254

7. Run main.sh to build namespace

chmod +x main.sh #allow permission to execute

./main.sh net

8. If didn’t work, may have to manually delete namespace and try again

ip netns del h1

ip netns del h2

Task 2: Define Protocol via Scapy:

1. Add the following code to ./src/Protocol.py

(Note: This and the following Sender.py and Receiver.py are python files, format is important, just copy and pasting code from the slides may result in problems.)

class Protocol(Packet):

# Set the name of protocol (Task 2.)

name = 'Student'

# Define the fields in protocol (Task 2.)

fields\_desc = [

StrField('index', '0'),

StrField('dept', 'cs', fmt='H', remain=0),

IntEnumField('gender', 2, {

1: 'female',

2: 'male'

}),

StrField('id', '000000', fmt='H', remain=0),

]

Task 3: Send Packets

1.Set own packet header in ./src/sender.py

# Set source and destination IP address (Task 3.)

src\_ip = '10.0.1.1'

dst\_ip = '10.0.1.2'

# Set source and destination port (Task 3.)

src\_port = 1024

dst\_port = 80

# Define IP header (Task 3.)

ip = IP(src=src\_ip, dst=dst\_ip)

# Define customized header (Task 3.)

my\_id = '0616020'

my\_dept = 'cs'

my\_gender = 'male'

student = Protocol(id=my\_id, dept=my\_dept, gender=my\_gender)

2.Set code to send packets in ./src/sender.py

# TCP connection - ACK (Task 3.)

ack = tcp\_syn\_ack.seq + 1

tcp\_ack = TCP(sport=src\_port, dport=dst\_port, flags='A', seq=1, ack=ack)

packet = ip / tcp\_ack

send(packet)

print '[INFO] Send ACK'

# Send packet with customized header (Task 3.)

ack = tcp\_ack.seq + 1

tcp = TCP(sport=src\_port, dport=dst\_port, flags='', seq=2, ack=ack)

packet = ip / tcp / student

send(packet)

print '[INFO] Send packet with customized header'

# Send packet with secret payload (Task 3.)

ack = tcp.seq + 1

tcp = TCP(sport=src\_port, dport=dst\_port, flags='', seq=3, ack=ack)

payload = Raw(secret[i])

packet = ip / tcp / payload

send(packet)

print '[INFO] Send packet with secret payload'

Task 4: Sniff Packets

1.Receive and sniff packets, add code to ./src/receiver.py

# Set source IP address and destination interface (Task 4.)

dst\_iface = 'h2-eth0'

src\_ip = '10.0.1.1'

# Sniff packets on destination interface (Task 4.)

print '[INFO] Sniff on %s' % dst\_iface

packets = sniff(iface=dst\_iface, prn=lambda x:packetHandler(x))

# Dump the sniffed packet into PCAP file (Task 4.)

print '[INFO] Write into PCAP file'

filename = './out/lab1\_0' + id + '.pcap'

wrpcap(filename, packets)

Task 5:Run sender and receiver

1.Open tmux with two panes, switch to the two namespaces in each pane.

# Open tmux

tmux

# Open new pane in horizontal

Ctrl-b

Shift-%

# Run namespace h1 in your left pane

./scripts/main.sh run h1

# Run namespace h2 in your right pane

./scripts/main.sh run h2

2.Run receiver first in h2

# Run receiver.py in h2, remember to switch over to the pane running namespace h2 first

python receiver.py

3.Run sender next in h1

# Run sender.py in h1, remember to switch over to the pane running namespace h1

python sender.py

4.If no problems, sender will succeed in sending all the packets and stop running, if stuck, check all the python codes(Protocol.py, sender.py, receiver.py) first.

5.Use tcpdump to show your PCAP file(remember to change to ./src/out first)

# Dump the PCAP via tcpdump

tcpdump -qns 0 -X -r lab1-0616020.pcap

Task 6: Push your files to remote

1.Push image to Dockerhub

# Create a new image from a container’s changes

docker commit cn2018\_c baili0411/cn2018\_lab1

# Login to your Docker registry

docker login

# Push an image to a registry

docker push baili0411/cn2018\_lab1

2.Push files to GitHub(Since we push this from the container, which doesn’t have the edited Docker file, remember to commit and push the Docker file from where we edited it)

# Get and set repository or global options

git config --global user.name "Baili Deng"

git config --global user.email "rayman0411@gmail.com"

# Add your files into staging area

git add .

# Commit your files

git commit –m "Commit lab1 in class"

# Set the remote URL to your remote repository

git remote set-url origin https://github.com/nctucn/lab1-baili0411.git

# Push your files to remote repository

git push origin master

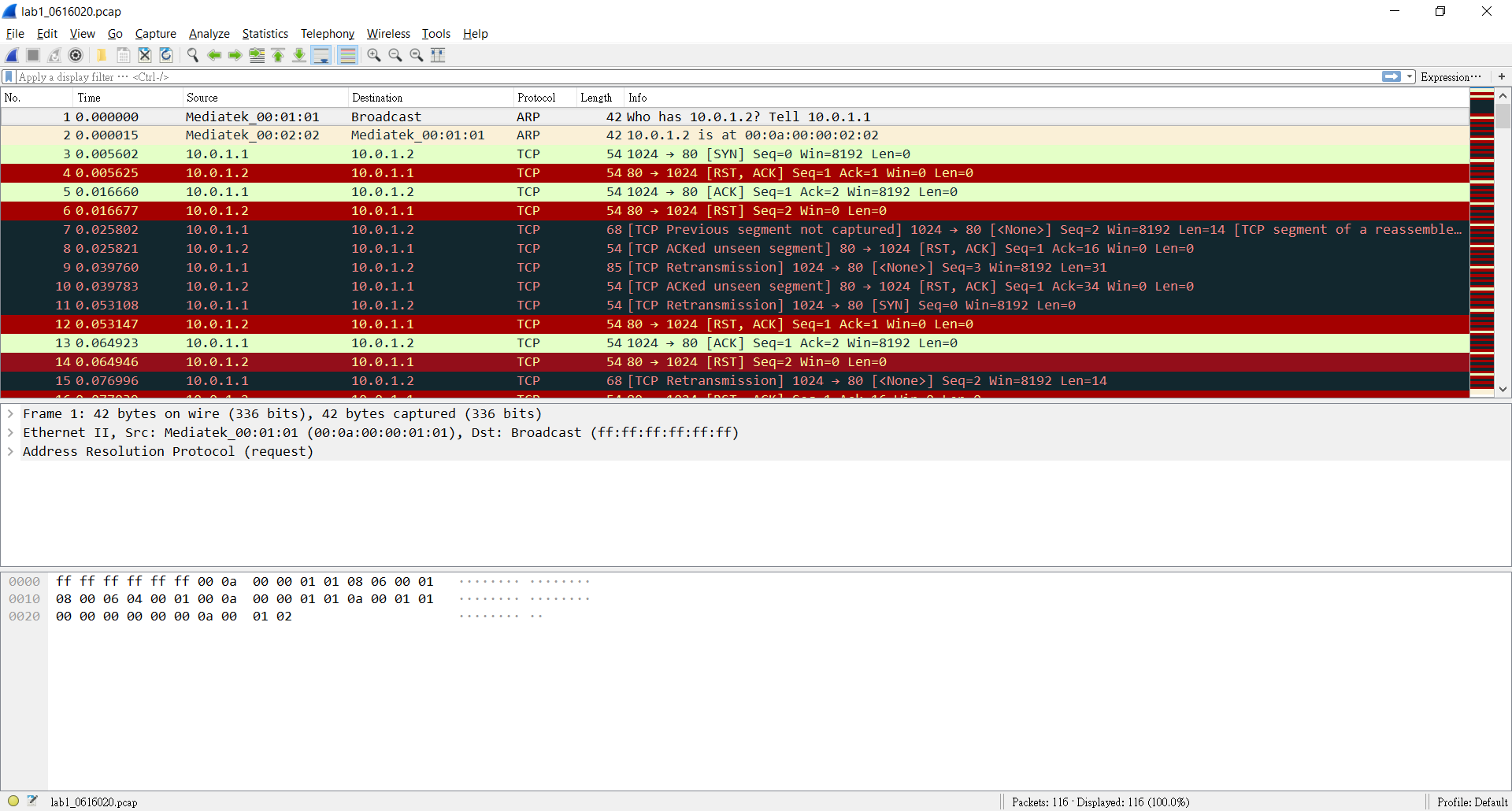
Task 7:Load PCAP via Wireshark

1.Download code and files from GitHub

git clone https://github.com/nctucn/lab1-baili0411.git

2.Install Wireshark

3.Load PCAP via Wireshark



(screenshot after loading the PCAP file)

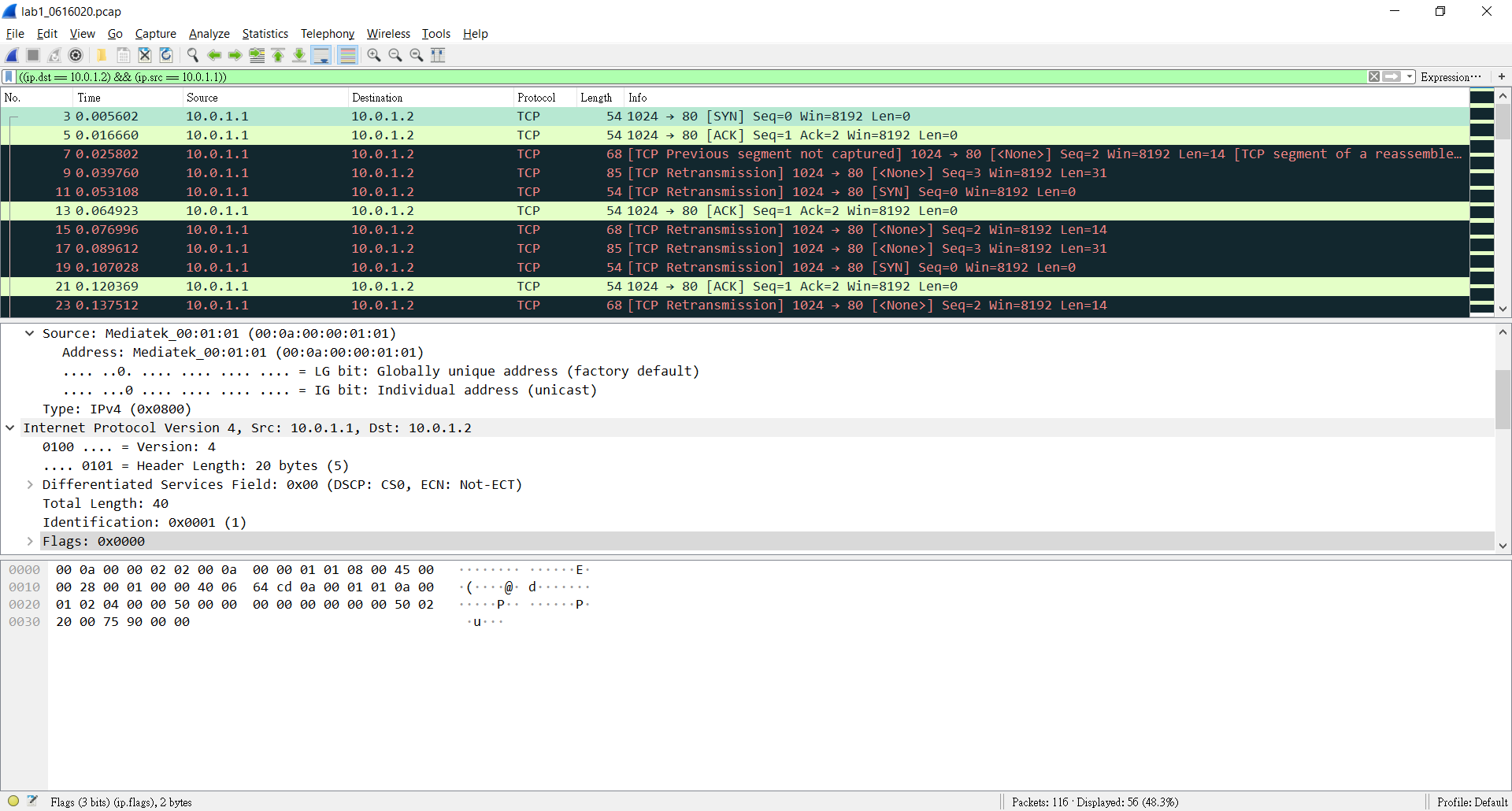
Task 8:Filter the target packet

1.Filter the packets of our defined protocol

Enter the filter command in DisplayFilter.

I filtered by the source and destination ip

((ip.dst == 10.0.1.2) && (ip.src == 10.0.1.1))



(screenshot after applying filter)

Judging by the Seq and ACK, these should be the packets of our defined protocol.

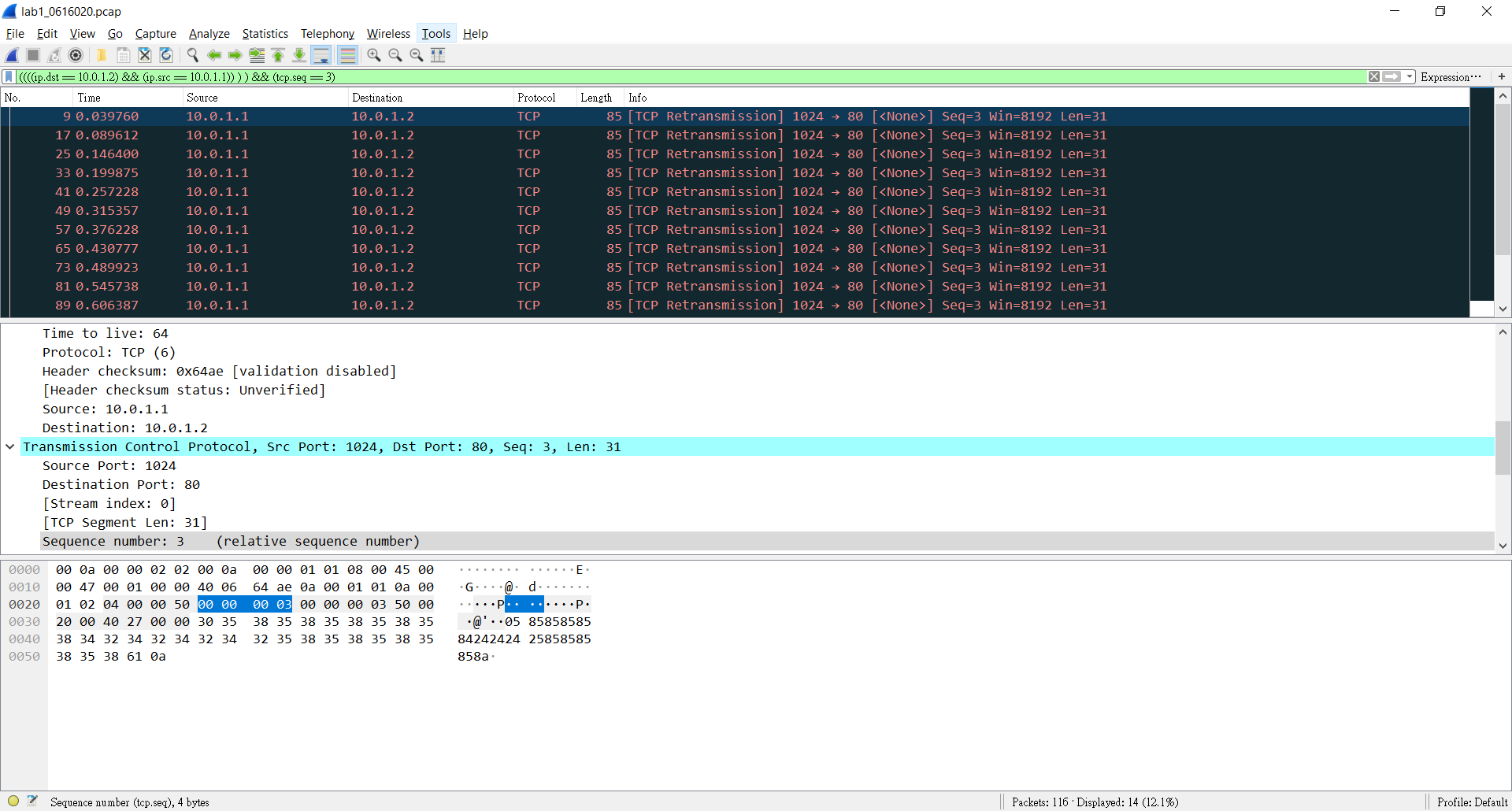
2.Filter the packets with the “secret” bits

Enter the command in the DisplayFilter.

According to sender.py, the packets with the secret bits have seq = 3, so we filter out the packets with seq = 3 from the packets with our protocol.

We use this filter command:

((((ip.dst == 10.0.1.2) && (ip.src == 10.0.1.1)) ) ) && (tcp.seq == 3)



Exactly 14 packets, this should be correct.

3.Get the secret key from the packets.

Since I didn’t know how to handle the data as seen from Wireshark to find the bits using a script, I just manually written down all the first bits from the 14 packets and formed the secret key.

My secret key is 02061600206160.

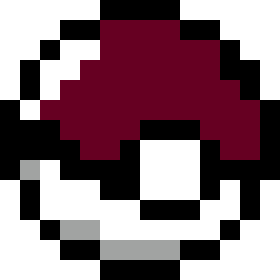
Task 9: Decode the secret key

1.Execute decoder.py with the argument being the secret key

python decoder.py 02061600206160

We should obtain a file related to Pokemon.

I got a Pokeball, so this should be correct.



(Image received in ./src/out)

Bonus:

What have I learned in this lab?

I learned how to set Wireshark DisplayFilter, how to setup a Docker image and container, and using GitHub in conjunction with Git. The rest (setting protocols, packets with custom headers, receiving and sniffing packets, Dockerfile settings, setting namespace) I probably need a bit more time to figure out, since the actual execution in the lab means that we mainly did those parts by copying certain parts code down, with other parts being already finished. I did understand some parts in sender.py, Protocol.py and receiver.py while debugging why it wasn’t working, mainly the overall logic, but not exactly what every line of code does.

What difficulty you have met in this lab?

The main difficulty I had in the lab is debugging what went wrong with my sender.py, receiver.py and Protocol.py. The code looked correct, but there was still an error message that kept popping up. I couldn’t find where I went wrong, and the TAs didn’t exactly help me realize where I went wrong, either. I decided to redo the whole thing again, this time by typing all the code myself instead of copying and pasting from the slides, then it worked. There was probably some format or syntax problem when I directly copied the code from the slide.

Another part I got stuck on is connecting to the docker image. The slides told us to connect to either 127.0.0.1 or 0.0.0.0, and both doesn’t work. Perhaps Docker on Windows (or Kitematic) works a bit differently, since we had to use find the IP address through Kitematic. It went smoothly once the TAs told us to find the address there.