NCTU CN2018 Lab. 1 – Packet Manipulation via Scapy

Student name: 王祥任 Student ID: 0616309 Department: CS

Task 1 – Environment setup

首先從助教的 GitHub 中取得必須檔案。

$git clone https://github.com/yungshenglu/Packet_Manipulation

從 Dockerfile 的修改做起。

$cd Packet_Manipulation/docker

$vim Dockerfile

在 Dockerfile 中加入以下指令 (在實作時出現第一個問題，最後詳談)。

Copy the following configuration to the Dockerfile (./docker/Dockerfile)

```
# 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
```

將執行介面轉為 WINDOWS 的 CMD(administration)，並將檔案路徑移向./docker，以建立名為 cn2018_c 的 container。

$cd docker

利用 Dockerfile 建立 image(cn2018)

$docker build -t cn2018 . (接著會跑出一段文字)

利用 cn2018 的 image 建立一個 container 叫 cn2018_c

$docker run -d -p 9487:22 –privileged –name cn2018_c cn2018 (同上)

設定 cn2018_c port 22

$docker port cn2018_c 22(記得會有一段流水號跳出)

至此我們需要的 container 建立完畢。接下來需要藉由 SSH 連線登入 Docker 的 container。

Windows 介面的情況下，在安裝完 PieTTY 後，接下來輸入

IP address : 127.0.0.1　Port: 9487

即可進入 container 中登入。

本來應該是這樣。但當天好像是因為 Windows 的問題，實際上的 IP 位址並不是 127.0.0.1。在這裡進度停滯了大概 1 小時後，我從 kitematic 中發現可以新增 ubuntu-env:16.04 的映像，同時也發現了在前面步驟建立的 cn2018_c 被新增至 kitematic 的列表中，而 IP 位址應為 192.xxx.x.x，Port 仍為 9487。在 PieTTY 中輸入後，總算是成功連入 cn2018_c 中。

以 root 登入 container 中。

Login: root　Password: cn2018

接下來轉移到 container 中 Packet_Manipulation 的位址，並導向下列路徑。

$cd src/scripts/

開啟 main.sh 並修改。

$vim main.sh

將以下指令打入 main.sh 中，以建立 h2's namespace。

Create the namespace in ./src/scripts/main.sh for h2 (i.e., receiver) (h1's namespace has been created)

```
# 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
```

修改完畢後，離開 vim，執行 main.sh 建立 namespace。

$chmod +x main.sh

$./main.sh net (接下來會跳出大量紀錄，注意有無錯誤)

Task 2– Define protocol via Scapy

這邊須注意一件事，從此開始修改的檔案都已經不是在本地電腦上，而是在 container 中。(故 container 中的 Dockerfile 其實還是原本未修改的)

以 vim 開啟在 src 中的 Protocol.py 並修改。

```
$vim Protocol.py
```

設定協議的名稱為”Student”。

設定協議的 fields 的預設值，包含”index” ,”dept” ,”gender” ,”id”。

以我個人的情況就是”0”, “cs”, “2”, “616309”

總而言之，照著備註複製貼上修改就行了。

Define your protocol: ID header format

- Copy the following code to ./src/Protocol.py

```
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

以 vim 開啟在 src 中的 sender.py 並修改。

```
$vim sender.py
```

設定來源 IP 位址為”10.0.1.1”，設定目的 IP 為址為”10.0.1.2”。

設定來源 PORT 為 1024，設定目的 PORT 為 80(HTTP)。

設定 IP header、設定自訂協議 header。

總而言之，照著備註複製貼上修改就行了。

Send packets:

Add the codes below 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'
```

在這邊有一件事情需要注意，觀察這些 CODE，可以導出 PART A 作業和 TASK 8 中需要打入的過濾指令。

Task 4-Sniff packets

以 vim 開啟 src 中的 receiver.py 並修改。

```
$vim receiver.py
```

設定來源 IP 位址和目的介面。

偵測封包並將封包 dump 為 pcap 檔，存放為./out/lab1_0616309.pcap 檔名中”616309”來自自訂協議 header 中的”id”。

Add the codes below in ./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

接下來就要開啟 h1,h2 兩個 namespace，並利用 sender.py 還有 receiver.py 來傳送和接收封包。

首先開啟 tmux。

$tmux

垂直分割畫面。

Ctrl+b Shift+%

在左側運行 namespace h1

Ctrl+b Arrow-left (移向左側視窗)

$ ./scripts/main.sh run h1 (注意路徑要留在 src 內)

在右側運行 namespace h2

Ctrl+b Arrow-right (移向右側視窗)

$ ./scripts/main.sh run h2 (注意路徑要留在 src 內)

在 h2 先執行 receiver.py (如果不先執行 receiver，就算先運行 sender 也沒有負責接收的程式)

Ctrl+b Arrow-right (移向右側視窗)

$ python receiver.py (注意路徑要留在 src 內)

然後在 h1 執行 sender.py 來傳送封包給 h2

Ctrl+b Arrow-left (移向左側視窗)

$ python sender.py (注意路徑要留在 src 內)

此時右側視窗的 h2 會開始接收大量封包，並顯示紀錄。結束之後，就可以利用 tcpdump 來顯示 pcap 檔的內容

$ tcpdump -qns 0 -X -r lab1_0616309.pcap

在接收完全部的封包後，便會得到 lab1_0616309.pcap 檔，還有一個 recv_secret.txt 檔(由 receiver.py 製作的)。

Task 6-Push your files to remote

將更改過後的 docker image 上傳至自己的 Docker Hub。

Push your image to Docker Hub

```
# Create a new image from a container's changes
$ docker commit cn2018_c <DOCKER_HUB_ID>/cn2018_lab1
# Login to your Docker registry
$ docker login
# Push an image to a registry
$ docker push <DOCKER_HUB_ID>/cn2018_lab1
```

其中 <DOCKER_HUB_ID> 即是前面提及的流水號，如果忘記了也沒關係，可以利用 docker 的指令查詢。

$ docker container ls

cn2018_c 那項即是我們需要的<DOCKER_HUB_ID>。

再來是將更改後的檔案上傳至 GitHub。

首先先將路徑變更到最上層的目錄。(假設目前在 src 中)

$ cd .. (直到回到 Packet_Manipulation 為止)

此步驟是為了方面讓以下的指令可以捕捉到全部的子目錄，包含 src 還有 docker，所以回到這兩個資料夾所在的目錄位置。

再來輸入以下指令將檔案上傳至 lab1-allen880117(我個人的情況)

Push your files to GitHub

```
# Get and set repository or global options
$ git config --global user.name "<NAME>"
$ git config --global user.email "<EMAIL>"
# 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-<YOUR_ID>.git
# Push your files to remote repository
$ git push origin master
```

<YOUR_ID> = allen880117

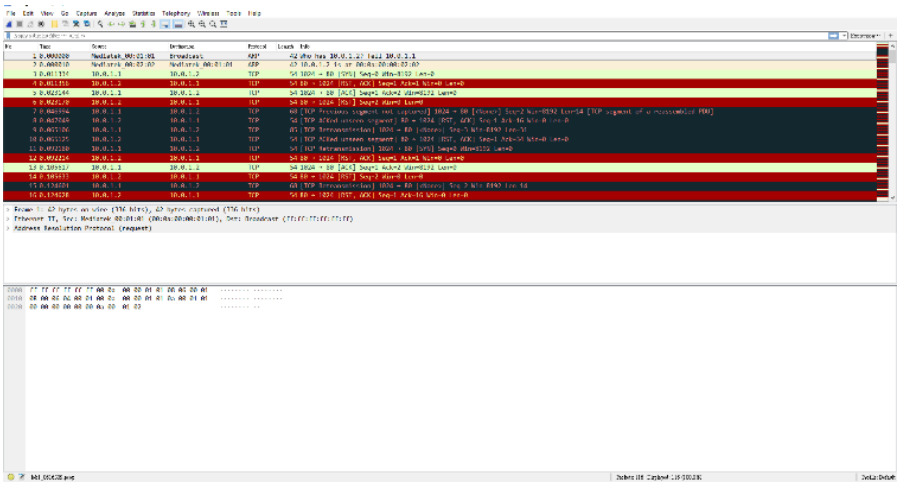
但是需要注意，在 container 中我們只對 src 底下的檔案有做修改，而最一開始修改的 Dockerfile 在 container 中其實還是修改前的版本。故記得之後要另外將修改過的 Dockerfile 重新上傳，做法基本上和前述相同。

Task 7-Load the packet trace via Wireshark

將自己的檔案從 GitHub 上複製下來(linux 指令)

```
$ git clone https://github.com/nctucn/lab1-allen880117.git
```

用 Wireshark 開啟./src/out/lab1_0616309.pcap



Task 8-Filter the target packet

接下來就是要分析封包，取得我們需要的資訊。

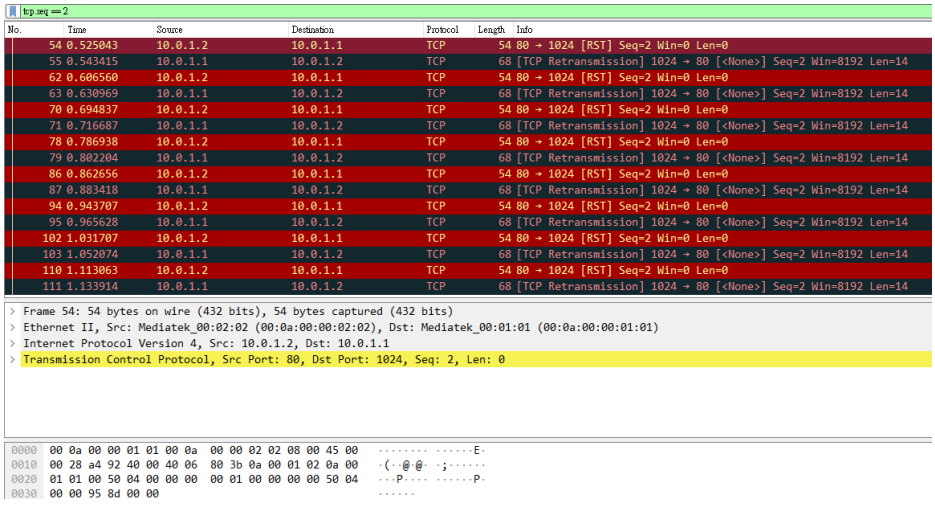
首先是找到那些有著我們自訂協議的封包，此處可以利用 wireshark 的 displayfilter，輸入指令即可以過濾封包。

根據我們在 Task 3 中對 sender.py 所新增的 code，我們可以發現包含著我們自訂協議"student"的封包，tcp 的 seq 為 2。

```
# 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'
```

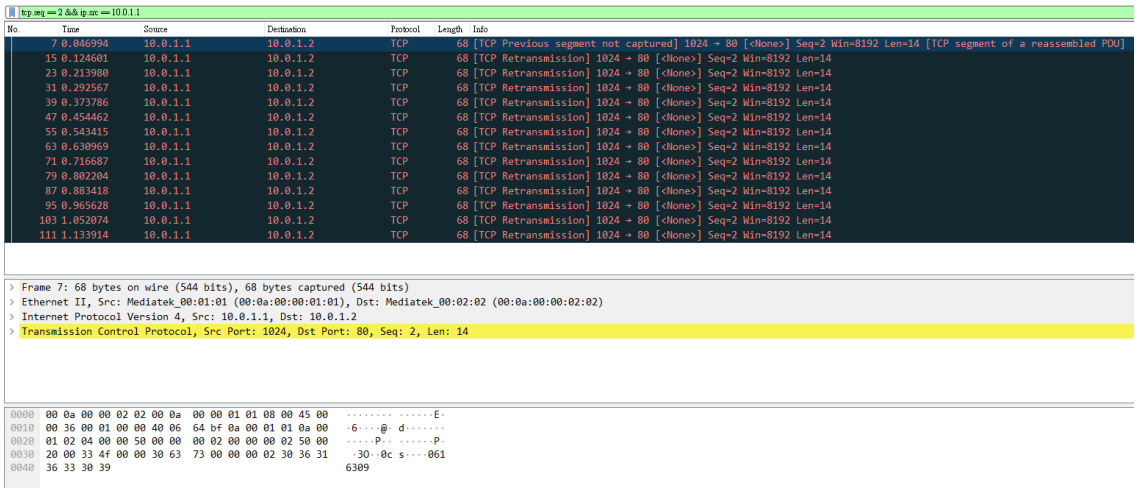
所以我們可以在 displayfilter 中輸入

```
tcp.seq == 2
```

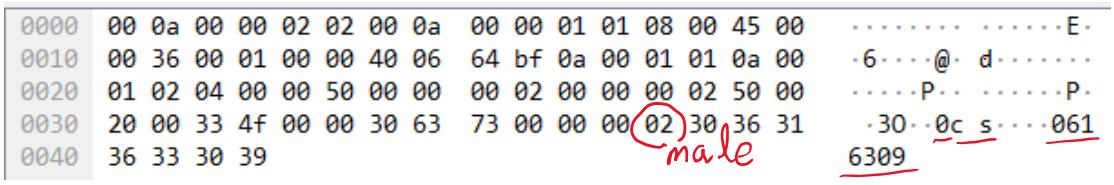


但只有這樣還不夠，裡面還參雜著從接收端(h2)回傳的封包，所以再添加一項指令，限定只需要來源 IP 為 10.0.1.1 的封包

tcp.seq == 2 && ip.src == 10.0.1.1



而在最下方欄位即可見到自訂協議的 header。



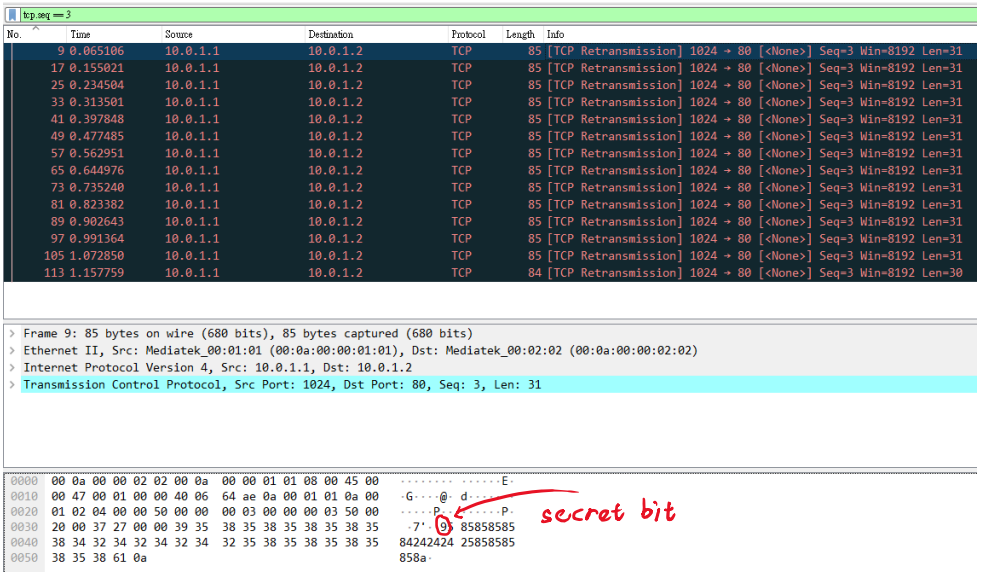
再來是從封包中找出”secret” bits。

同樣是從 Task 3 中 sender.py 所新增的 code 可以發現，包含 secret payload 的封包，其 tcp 的 seq 皆為 3。

```
# 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'
```

所以我們可以在 displayfilter 中輸入

tcp.seq == 3



將這 14 個封包所包含的 secret bits 全部依封包編號組合在一起，即可得到一個 14-digit “secret” key。

我的 key 是 90361609036160。

Task 9- Decode the secret key

接下來就要利用./src/decoder.py 和在 Task 8 中取得的 14-digit “secret” key，來解密圖像。

注意執行 decoder.py 需要 pillow(PIL)套件，記得安裝。

```
$ pip install pillow (如果沒有安裝的話)
$ python decoder.py 90361609036160
```

```
wangxr@DESKTOP-IJHBADU:~$ cd lab1-allen880117/
wangxr@DESKTOP-IJHBADU:~/lab1-allen880117$ cd src
wangxr@DESKTOP-IJHBADU:~/lab1-allen880117/src$ python decoder.py 90361609036160
python: can't open file 'decoder.py': [Errno 13] Permission denied
wangxr@DESKTOP-IJHBADU:~/lab1-allen880117/src$ sudo python decoder.py 90361609036160
[sudo] password for wangxr:
[INFO] Your key is 90361609036160
[INFO] Decode successful
[INFO] Finish decoding
```

因為這個帳號不是 root，所以需要 sudo。

成功解密後，在./src/out 裡便會產生一個 lab1_0616309.png。

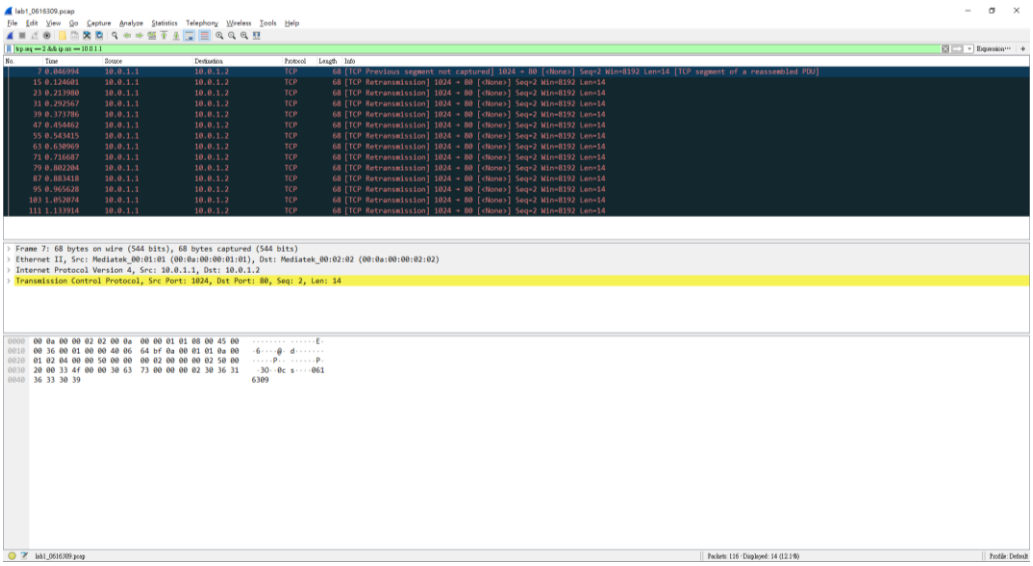
看來只是一個生灰塵的モンスターボール

Task 10- Report

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

   ANS : tcp.seq == 2 && ip.src == 10.0.1.1

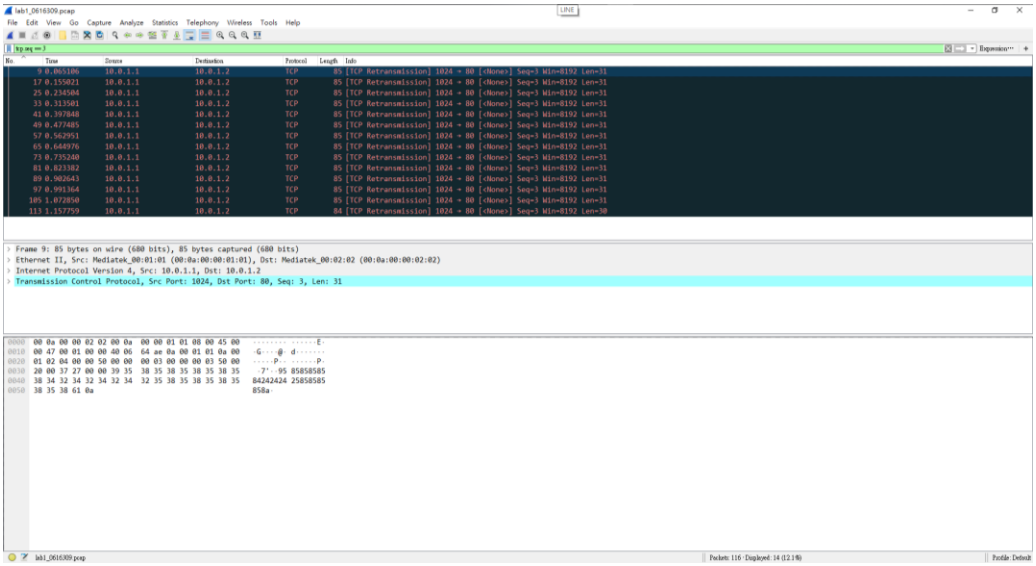
2. Show the screenshot of filtering the packet with customized header.



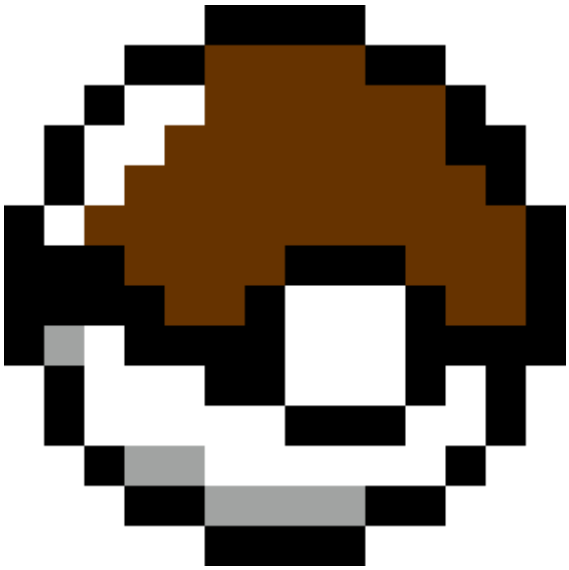
3. What is your command to filter the packet with “secret” payload on Wireshark?

   ANS : tcp.seq == 3

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



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



Bonus

1. What you have learned in this lab ?

除了對封包傳遞有多一點理解、稍微會使用 wireshark 去分析封包以外，學最多的還是在 linux 系統下的操作和 Git 的使用，這次 Lab 中是我第一次使用 Git 來對檔案做管理和更新，花了不少時間去理解指令和上手，不過確實是挺好用的。

2. What difficulty you have met in this lab?

其實在這次的 Lab 中，碰到過蠻多問題的。一開始，lab1_tasks.pdf 最早給的指令其實有錯，包括 ：

1. Ubuntu-env:16.0 應為 ubuntu-env:16.04
2. RUN apt-get install -y tcpdump 少了 -y
3. Clone 的網址為個人的 lab1-<GithubID>

不過這還不算太大的問題，卡最久的部分是用 SSH 連上 Docker 的 Container，粗估卡了一個小時多，因為說明文件中所給的 IP 位址一直被拒絕存取，助教認為可能是 Windows 的問題。後來直接用 kitematic 建立新的 container，ubuntu-env:16:04 的 image 已經存在，開啟這個 image 後，cn2018_c 也出現在了列表中，而它的 IP 位址是 192.xxx.x.x，嘗試用此 IP 連結後，才終於連上線，得以繼續完成 LAB。