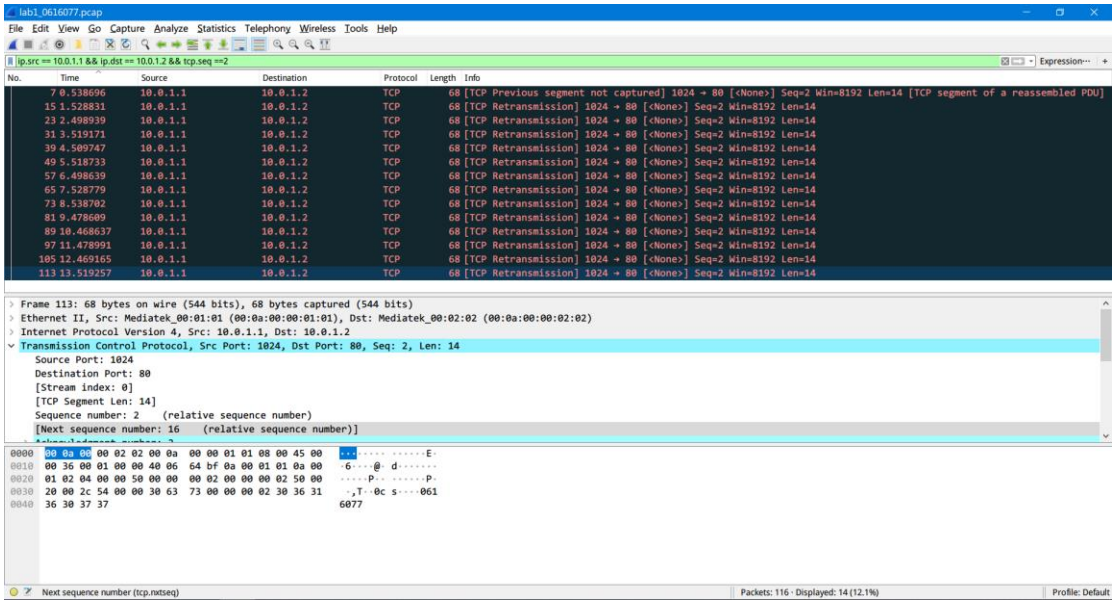
NCTU CN2018 Lab. 1 – Packet Manipulation via Scapy

Student:李柏漢 0616077 CS

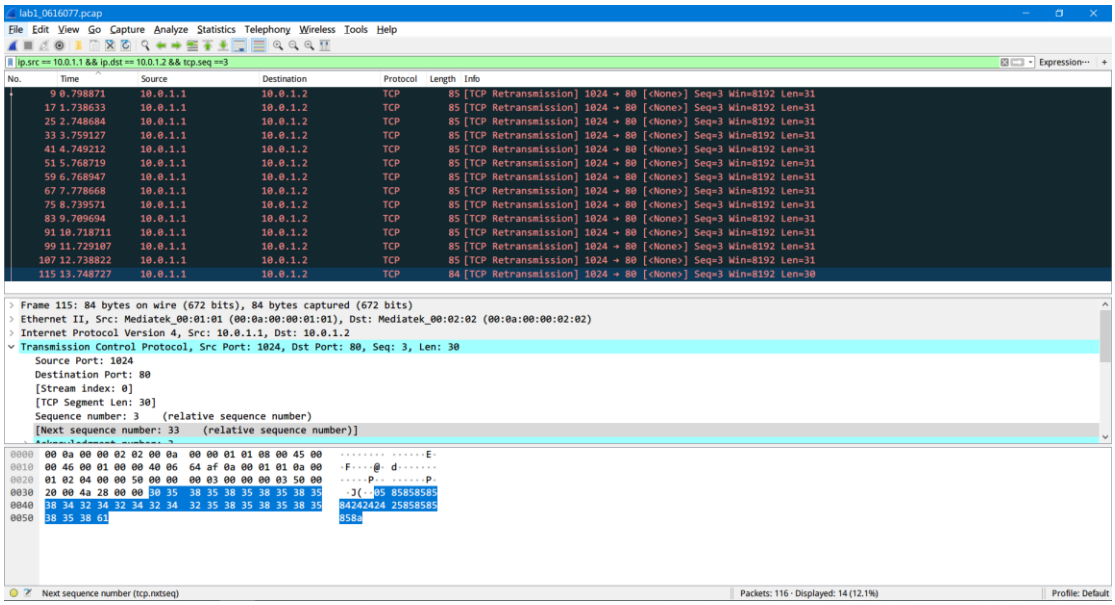
1. **What is your command to filter the packet with customized header on Wireshark?**

   ip.src == 10.0.1.1 && ip.dst == 10.0.1.2 && tcp.seq == 2



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

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



3. **Show the result after decoding the “secret” payload.**

lab1_0616077.png

Color of the Poké Ball would depend on id[3:-1](‘#’+id[3:-1])

Describe each step in this lab in detail:

Task 1 – Environmental setup

先 clone 一份 yungshenlu/Packet_Manipulation 到本機

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

編輯 Dockerfile

從 yungshenglu/ubuntu-env:16.04 下載 base image (apt-get -y means always answers yes)

```
FROM yungshenglu/ubuntu-env:16.04
```

更新 software repository 並安裝 tcpdump 和 scapy

```
RUN apt-get update
RUN apt-get -y install tcpdump
RUN pip install scapy
```

Use argument to assign passwd and permit root login

```
RUN echo 'root:cn2018' | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-
password/PermitRootLogin yes/' /etc/ssh/sshd_config
```

SSH login fix. Otherwise user is kicked off after login

```
RUN sed 's@session\s*required\s*pam_loginuid.so@session
optional pam_loginuid.so@g' -i /etc/pam.d/sshd
```

Set the envionment variables

```
ENV NOTVISIBLE "in users profile"
ENV LC_ALL C
RUN echo "export VISIBLE=now" >> /etc/profile
```

Set the container listens on the specified ports(22 for ssh) at runtime

```
EXPOSE 22
```

Set the entrypoint

```
CMD ["/usr/sbin/sshd", "-D"]
```

在 win10 上裝 docker(要專業版)，開啟 docker，用 Dockerfile build container

先 cd 到./docker，build image from Dockerfile

```
docker build -t cn2018 .
```

再從 cn2018 這個 image build 名為 cn2018_c 的 container

-d means detach(run container in background & print container ID)

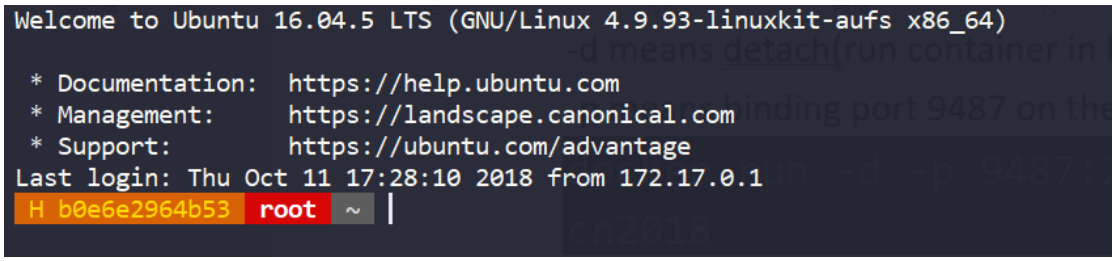
-p means binding port 9487 on the host to port 22 in the container

```
docker run -d -p 9487:22 --privileged --name cn2018_c cn2018
```

從 cmd 以 root 身分從 9487 這個 port 進入 cn2018_c (密碼 cn2018

```
$ ssh -p 9487 root@127.0.0.1
```

成功畫面



在 docker 內進行 git clone 和一些設定(username,email,remote set-url origin)

然後先 git push origin master，把最原始的檔案 push 上去

```
$ git clone
https://github.com/yungshenglu/Packet_Manipulation
$ git config --global user.name "your_name"
$ git config --global user.email "your_email"
$ git remote set-url origin
https://github.com/nctucn/lab1-0616077.git
$ git push origin master
```

./src/scripts/main.sh 中建立 namespace h2 所需的 command(h1 已經寫好

Create & Delete h2 network namespaces

```
ip netns add h2
```

```
ip netns del h2
```

Bring up the lookup interface in h2

```
ip netns exec h2 ip link set lo up
```

Set the interface of h2 to h2-eth0 & Delete the interface of h2-eth0

```
ip link set h2-eth0 netns h2
ip link delete h2-eth0
```

Activate h2-eth0 and assign IP address

```
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

```
ip netns exec h2 sysctl net.ipv6.conf.h2-eth0.disable_ipv6=1
```

Set the gateway of h2 to 10.0.1.254

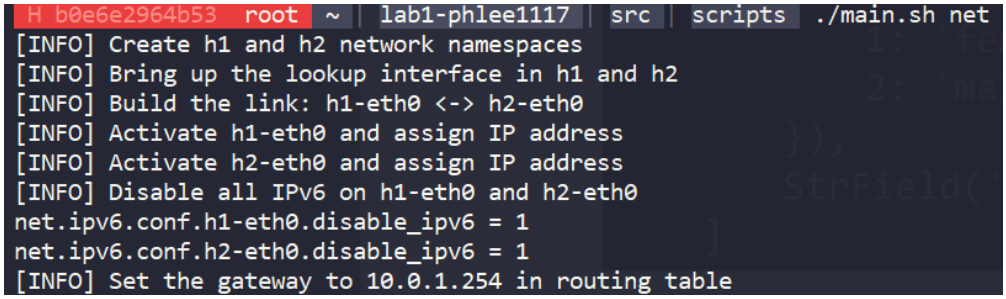
```
ip netns exec h2 ip route add default via 10.0.1.254
```

Run `./src/scripts/main.sh` net to build the namespace

chmod +x 增加執行權限

```
$ cd ./src/scripts
$ chmod +x main.sh
$ ./main.sh net
```

成功畫面:



Task 2. Define protocol via Scapy in ./src/Protocol.py

自定義名為 Student 的 Protocol(index,dept(default=cs),gender,id)，並將其合併(bind)到 TCP 的 Protocol 上

```
class Protocol(Packet):
    # Set the name of protcol (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

設定傳送的封包

來源&目的地 ip、port

```
src_ip = '10.0.1.1'
dst_ip = '10.0.1.2'
src_port = 1024
dst_port = 80
```

設定屬於自己的 IP & TCP customized header

```
ip = IP(src = src_ip, dst = dst_ip)
my_id = '0616077'
my_dept = 'cs'
my_gender = 'male'
student = Protocol(id = my_id, dept = my_dept, gender =
my_gender)
```

設定要傳送的封包 SYN、ACK、customized header、secret payload)

首先是 TCP 的 three-way handshake 中的 ACK(SYN 和 SYN_ACK 已完成

```
print '[INFO] Send ACK'
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)
```

再來是 customized header

```
print '[INFO] Send packet with customized header'
ack = tcp_ack.seq + 1
tcp = TCP(sport=src_port, dport=dst_port,
          flags='', seq=2, ack=ack)
packet = ip / tcp / student
```

```
send(packet)
```
最後是 secrey payload
```
print '[INFO] Send packet with secret payload'
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)
```
因為 len(secret)是 14 個，所以這四種封包會傳 14 次

Task 4. Sniff packets in ./src/receiver.py

設定來源 ip、目的地網路介面
```
dst_iface = 'h2-eth0'
src_ip = '10.0.1.1'
```
設定 Sniff 的回傳函數 packetHandler(使用 lambda 函數)，其中包含從 customized header 中存取 id、把 secret payload 存到 received 陣列中
```
print '[INFO] Sniff on %s' % dst_iface
packets = sniff(iface = dst_iface, prn = lambda x :
packetHandler(x))
```
(因為沒有設定何時結束，所以執行 python 時要按 ctrl+c 終止 sniff)

將 sniff 到的 packets 寫入./out/lab1_0616077.pcap
```
print '[INFO] Write into PCAP file'
filename = './out/lab1_0' + id + '.pcap'
wrpcap(filename, packets)
```
把 received 陣列元素逐行存入./out/recv_secret.txt
```
with open('./out/recv_secret.txt', 'w') as file:
    for line in received:
        file.write('%s' % line)
```

Task 5. Run sender and receiver

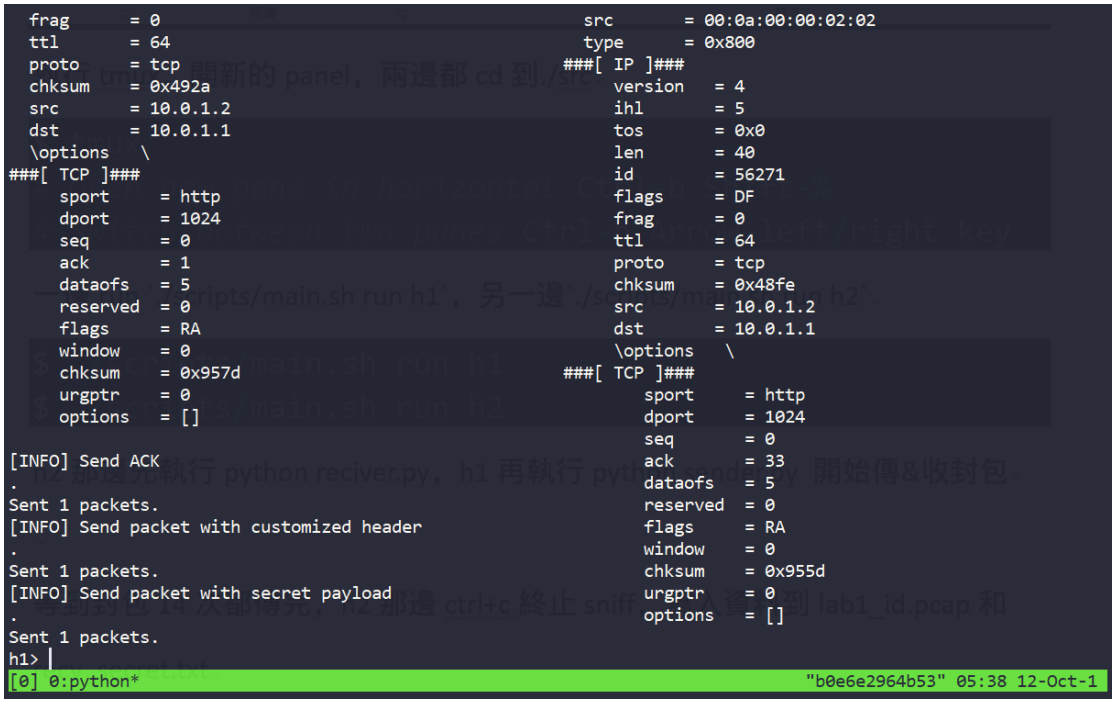
./src 底下要先新建 out 資料夾(不然 receiver.py 無法輸出到檔案(file not existed
```
$ cd ./src
$ mkdir out
```
執行 tmux，開新的 panel，兩邊都 cd 到./src
```
$ tmux
```

```
# Open new pane in horizontal Ctrl-b Shift-%
# Switch between two panes Ctrl-b Arrow-left/right key
```

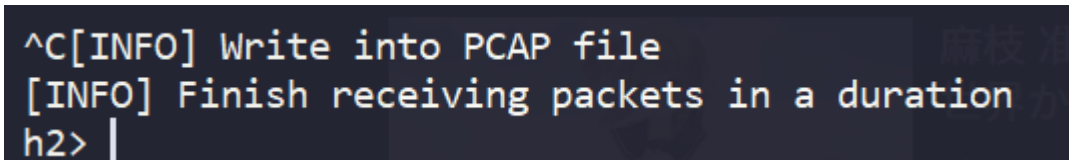
一邊 run `./scripts/main.sh run h1`，另一邊`./scripts/main.sh run h2`

```
$ ./scripts/main.sh run h1
$ ./scripts/main.sh run h2
```

h2 那邊先執行 python reciver.py，h1 再執行 python sender.py 開始傳&收封包



等到封包 14 次都傳完，h2 那邊 ctrl+c 終止 sniff，寫入資料到 lab1_id.pcap 和 recv_secret.txt



Use tcpdump to show your PCAP file

```
$ tcpdump -qns 0 -X -r lab1_phlee1117.pcap
```

Task 6 Push your files to remote

確認 lab1_id.pcap(using tcpdump or wireshark)和 recv_secret.txt 都 ok

回到本機 commit 這個 docker image 到 dockerhub_id/cn2018_lab1(可能需要 docker login

```
$ docker commit cn2018_c phlee1117/cn2018_lab1
$ docker login
$ docker push phlee1117/cn2018_lab1
```

一切咚咚都 git commit 完了，就 git push origin master

```
$ git add .
$ git commit -m "Finish lab1”
$ git push origin master
```

Task 7. Load PCAP via Wireshark

回本機把 git repository clone 下來

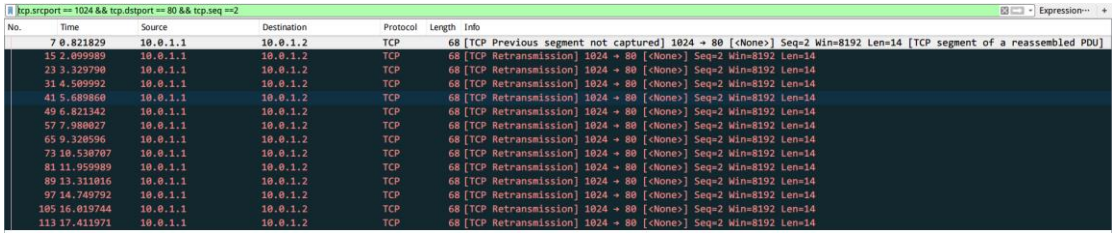
```
$ git clone
https://github.com/nctucn/lab1-phlee1117.git
```

安裝 wireshark，用 wireshark 打開 lab1_id.pcap

Task 8. Filter the target packet

Filter the packets of our defined protocol(customized header)

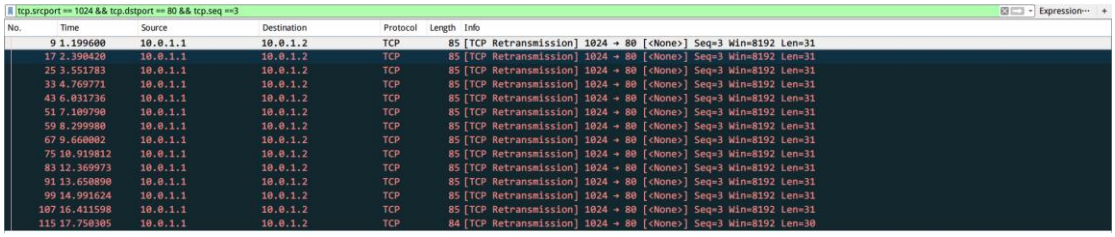
Filter Rule: tcp.srcport == 1024 && tcp.dstport == 80 && tcp.seq ==2



No.	Time	Source	Destination	Protocol	Length	Info
7	0.821829	10.0.1.1	10.0.1.2	TCP	68	[TCP Previous segment not captured] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14 [TCP segment of a reassembled PDU]
15	2.099989	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
23	3.329790	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
31	4.509992	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
41	5.689860	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
49	6.821342	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
57	7.980027	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
65	9.320596	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
73	10.530707	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
81	11.959989	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
89	13.311016	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
97	14.749792	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
105	16.019744	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14
113	17.411971	10.0.1.1	10.0.1.2	TCP	68	[TCP Retransmission] 1024 → 80 [<None>] Seq=2 Win=8192 Len=14

Filter the packets with the “secret” bits

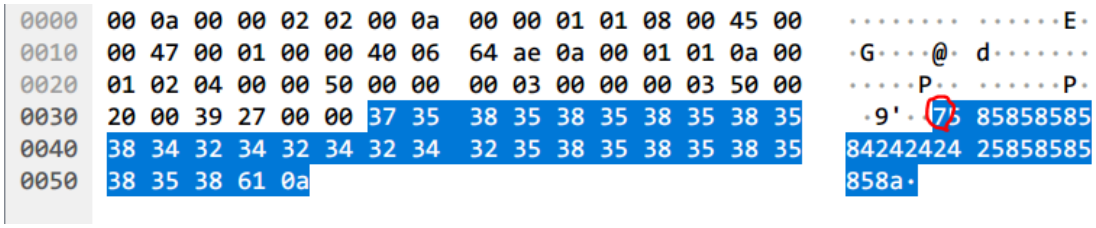
Filter Rule: tcp.srcport == 1024 && tcp.dstport == 80 && tcp.seq ==3



No.	Time	Source	Destination	Protocol	Length	Info
9	1.199600	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
17	2.398420	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
25	3.551783	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
33	4.769771	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
43	6.031736	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
51	7.109790	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
59	8.299980	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
67	9.660002	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
75	10.919812	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
83	12.369973	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
91	13.650890	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
99	14.991624	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
107	16.411598	10.0.1.1	10.0.1.2	TCP	85	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=31
115	17.750305	10.0.1.1	10.0.1.2	TCP	84	[TCP Retransmission] 1024 → 80 [<None>] Seq=3 Win=8192 Len=30

What is my secret key? How to find it?

由 secret payload 的 14 個的第一個 bit 發現 key 的秘密



```
0000  00 0a 00 00 02 02 00 0a  00 00 01 01 08 00 45 00  ........ ......E.
0010  00 47 00 01 00 00 40 06  64 ae 0a 00 01 01 0a 00  .G....@. d.......
0020  01 02 04 00 00 50 00 00  00 03 00 00 00 03 50 00  .....P.. ......P.
0030  20 00 39 27 00 00 37 35  38 35 38 35 38 35 38 35   .9'..75 85858585
0040  38 34 32 34 32 34 32 34  32 35 38 35 38 35 38 35  84242424 25858585
0050  38 35 38 61 0a                                    858a.
```

Task 9. Decode the secret key

到./src/內，python decoder.py key(key 為學號倒過來寫兩遍 總共 14 個數字 就會有屬於你的寶貝球出現在./src/out 了 讚

```
$ python decoder.py 77061607706160
[INFO] Your key is 77061607706160
[INFO] Decode successful
[INFO] Finish decoding
```

Bonus:

What you have learned in this lab?

首先是 docker 的運用，之前只有聽說過他的強大，實際使用後真的覺得是神器，一個 ubuntu 的環境就這樣在幾行 code 執行之下就能拿來使用。
Python 和 shell script 的一些運用也從助教的 code 中學習到很多。
Git 也是首次真的 commit，透過一堂課學習 git 好像有點遜，不過還是感謝有這次 lab 才能讓我稍微學會 Git 的一些觀念 XDD
tmux 也是第一次用，以前都覺得直接在 Terminal 新開分頁就好，切換起來也不慢，不過 tmux 能做的事似乎更多元(在同個 session 裡做事之類的。
TCP 的 Three-way Handshake 也從 sender.py 中學習到(SYN,SYN+ACK,ACK)
decoder.py 中 ImageColor.getrgb('#'+key)如果 key 是 3 個字，例如 123，則 rgb 解讀成#112233，如果 key 是 4 個字(如同這個 lab 的設計)，則會多出一個 alpha 值，代表不透明度。
後記:助教將 decoder.py 中的 key 改成 3 個字而已了，變成正常的 rgb 三個值，讓大家的 poke ball 顏色比較多變一點(原本資工系幾乎都綠的，現在從咖啡到紫色都有)

What difficulty you have met in this lab?

雖然都是 copy paste 居多，不過因為對 git 和 docker 不熟所以在 debug 過程中學到蠻多關於 commit 的觀念。
第一次嘗試 decoder.py 發現失敗了，印出一個 lab1_.png 的棋盤，因此研究了 decoder.py 和 receiver.py 的 code，發現在 receiver.py 內多了一行輸出 id 到 recv_secret.txt，讓解密失敗，刪掉該行 code 即可。
Demo 的時候某助教說要能在沒有 run namespace(h1 h2)的情形下 ping 到 h1 h2，我弄很久想破頭都不知道怎麼辦，沒有 run 怎麼 ping 呢…後來找另一個助教表示其實要在 run 的時候才去 ping，幸好(非抱怨。
Report 比想像中的累人，要一直 copy&paste…不過也在過程中了解每行 code 的細節和意義。