

网络与信息安全课内实验 2： DDos 攻击实验报告

班级：计算机 2101 班

姓名：田濡豪

学号：2203113234

1 实验平台及环境

本次实验使用个人计算机完成。

对于需要 Ubuntu 图形化界面或需要多台虚拟机同时运行的实验步骤，选择使用 Docker 进行容器化部署。Docker 镜像可以方便的进行修改、销毁、容器并发，性能高于不同虚拟机且不影响宿主机，为实验提供了高灵活性和容错空间。所选用的镜像版本为：

`kasmweb/desktop:1.16.1-rolling-weekly`

其余步骤使用 WSL（Windows Subsystem for Linux）完成，所安装的 Ubuntu 系统版本为：

`Ubuntu 22.04.4 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)`

对于每个实验步骤的具体环境配置，将在对应章节说明。

2 实验原理及目的

1. 安装虚拟机平台
2. 安装虚拟机
3. 在 Server 虚拟机上安装 apache 服务器
4. SYN-Flood 攻击
5. 解释 TCP 协议参数

3 实验过程

3.1 安装虚拟机平台

使用 Docker Desktop 安装 `kasmweb/desktop:1.16.1-rolling-weekly` 镜像，随后根据 KasmWeb 使用说明运行容器并将其暴露到 `https://127.0.0.1:6901/`:

```
sudo docker run --rm -it --shm-size=512m -p 6901:6901 -e VNC_PW=password  
kasmweb/desktop:1.16.0
```

3.2 安装虚拟机

作为最小体积的基础镜像，`kasmweb/desktop:1.16.1-rolling-weekly` 并没有携带实验用的基础软件，并且不开放管理员权限。为了满足实验要求，基于该基础镜像使用 Docker 工具进一步构建本次实验所用的镜像。特别的，由于实验 1 中已经有构建基础开发环境的 Docker 脚本。本次实验中的镜像基于上一次实验继续构建。

```
#Dockerfile  
#sudo as root role.  
  
FROM kasmweb/desktop:1.16.1-rolling-weekly  
USER root  
  
ENV HOME=/home/kasm-default-profile  
ENV STARTUPDIR=/dockerstartup  
ENV INST_SCRIPTS=$STARTUPDIR/install  
WORKDIR $HOME  
  
##### Customize Container Here #####  
ENV TZ=Asia/Shanghai  
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone \  
    && apt-get update \  
    && apt-get install -y sudo \  
    && echo 'kasm-user ALL=(ALL) NOPASSWD: ALL' >> /etc/sudoers \  
    && rm -rf /var/lib/apt/list/* \  
    && chown 1000:0 $HOME \  
    && $STARTUPDIR/set_user_permission.sh $HOME \  
    && sudo apt-get install -y gedit \  
    && sudo apt install -y iputils-ping \  
    && sudo apt-get install -y bind9 \  
    && sudo apt-get install -y dnsutils \  
    && sudo rndc-confgen -a \  
    # apache2 for experiment 2  
    && sudo apt-get install -y apache2 \  
    # wireshark for experiment 2  
    && apt install -y wireshark \  
    && sudo apt-get install -y python3-pip python3-dev \  
    && sudo echo "export PATH=\"`python3 -m site --user-base`/bin:~/.local/bin:$PATH\"" >>  
~/.bashrc \  
    && exec bash \  

```

```
&& source ~/.bashrc
```

```
##### End Customizations #####
```

```
ENV HOME=/home/kasm-user
```

```
WORKDIR $HOME
```

```
RUN mkdir -p $HOME && chown -R 1000:0 $HOME
```

```
USER 1000
```

默认情况下，Docker 所创建的容器实例处于桥接模式，因此属于同一局域网网段。使用 Shell 脚本运行两个容器并确认其局域网 IP 地址：

```
# check if sudo
```

```
if [ $(id -u) -ne 0 ]; then
    echo "Please run as root"
    exit 1
fi
```

```
# run and connect containers
```

```
# expose port 6901 for server
```

```
docker run --name server --cap-add=NET_ADMIN --privileged --rm -d --shm-size=512m
-p 6901:6901 -e VNC_PW=password xjtuns_exp2:latest
```

```
# expose port 6902 for client in a new terminal
```

```
docker run --name client --cap-add=NET_ADMIN --privileged --rm -d --shm-size=512m
-p 6902:6901 -e VNC_PW=password xjtuns_exp2:latest
```

```
# show container ip
```

```
echo "Server IP: "
```

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
server
```

```
echo "Client IP: "
```

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
client
```

```
# ask for execution completion
```

```
echo "Press any key to continue..."
```

```
read -p "Press any key to continue..." key
```

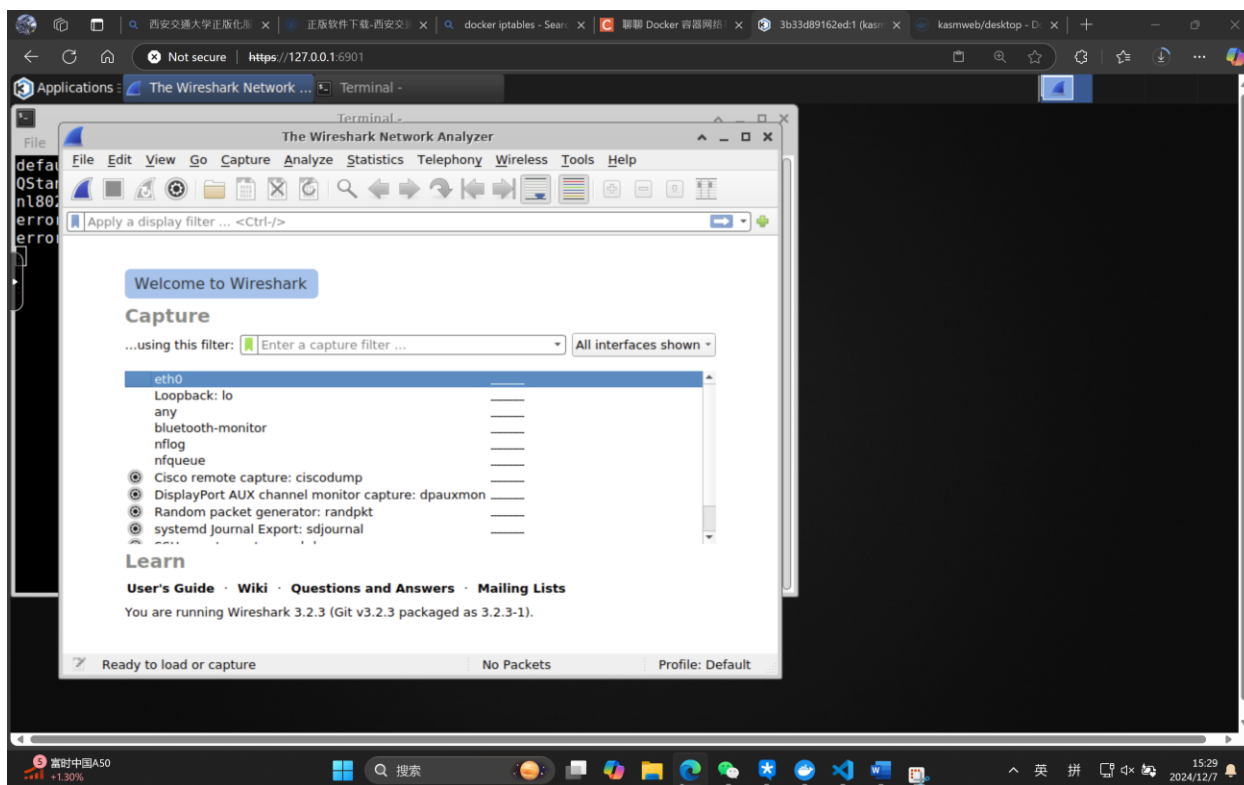
```
# stop and remove containers
```

```
docker stop server client # dns
```

输出如下：

```
(xjtu_ns) (base) ruhaotian@Book3Ultra-Ruhao:~/docker_learn$ sudo sh xjtu_ws_exp2.sh
77c949801c94f3e5c8c366398457f014e97a715def530d9921a33baa7c8626a0
4b49324c14734f62162edcde18f437188d2c2c0fb51756ded39f38a7b84d1b1c
Server IP:
172.17.0.2
Client IP:
172.17.0.3
Press any key to continue...
Press any key to continue...
```

可见 Server 和 Client（攻击者）的局域网 IP 分别是 172.17.0.2 和 172.17.0.3。安装后发现无法正常访问虚拟机。经过排查原因是上一次实验中为关闭防火墙仅用了 Docker 的 iptables，导致自动网路配置没有进行。重新开启后输入默认用户名和密码即可连接并使用虚拟机：



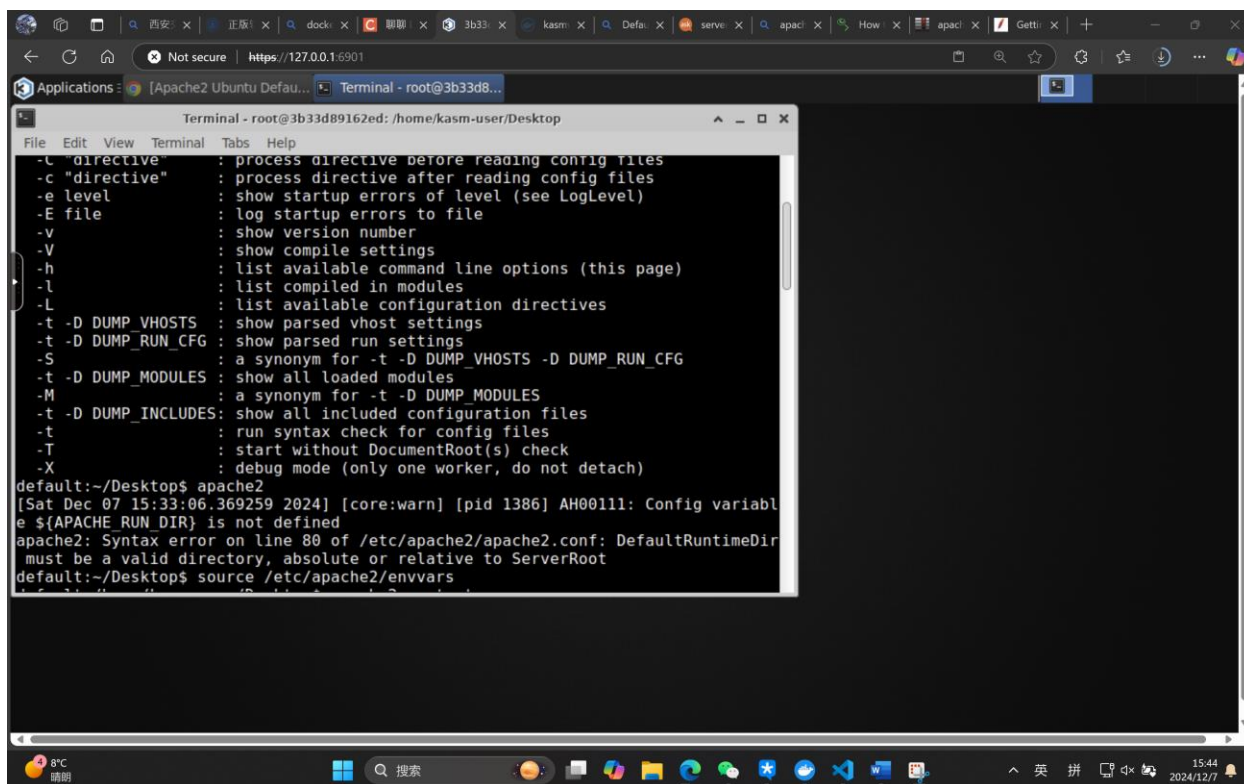
可见在镜像构建时预装的 Wireshark 软件也运行正常。

3.3 在 SERVER 虚拟机上安装 APACHE 服务器

在镜像构建时已经预装了 Apache 服务器。尝试直接访问 127.0.0.1，未能成功。随后发现 apache2 服务没有正确运行：

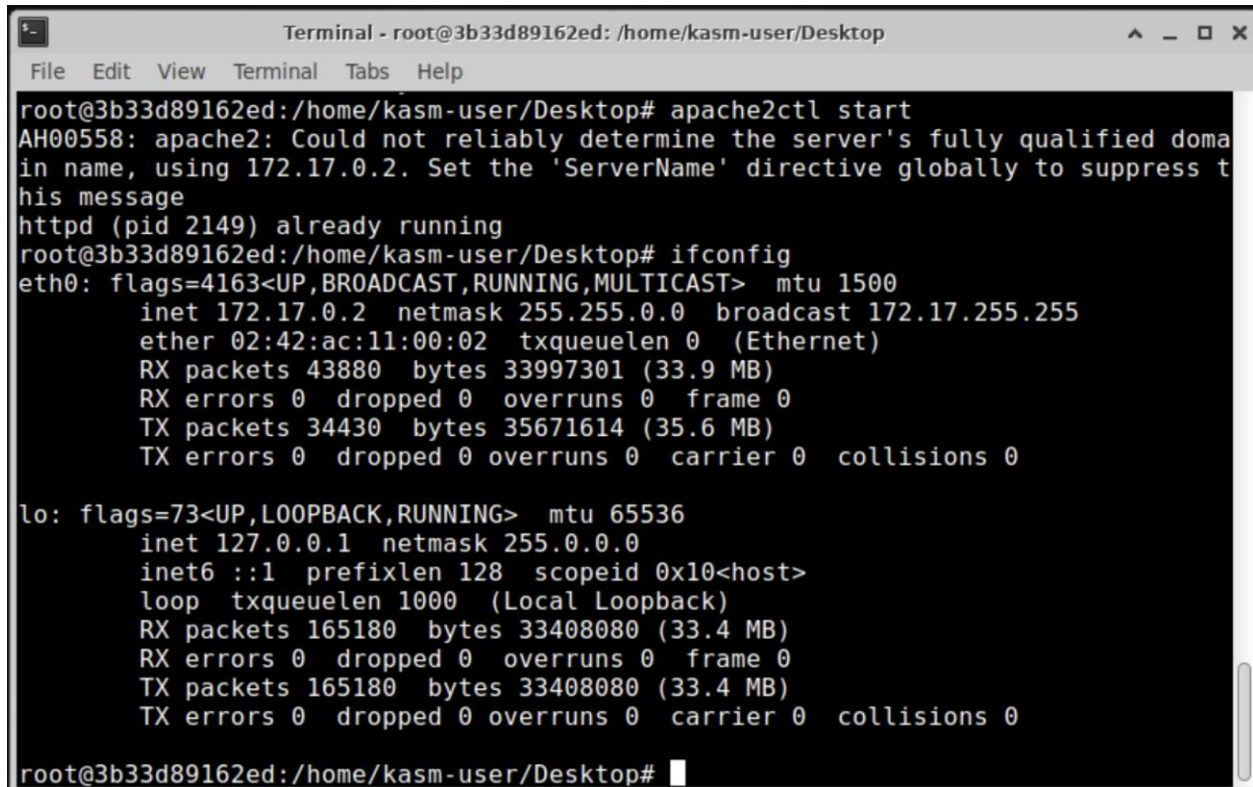
```
Terminal - root@3b33d89162ed: /home/kasm-user/Desktop
File Edit View Terminal Tabs Help
-C "directive" : process directive before reading config files
-c "directive" : process directive after reading config files
-e level : show startup errors of level (see LogLevel)
-E file : log startup errors to file
-v : show version number
-V : show compile settings
-h : list available command line options (this page)
-l : list compiled in modules
-L : list available configuration directives
-t -D DUMP_VHOSTS : show parsed vhost settings
-t -D DUMP_RUN_CFG : show parsed run settings
-S : a synonym for -t -D DUMP_VHOSTS -D DUMP_RUN_CFG
-t -D DUMP_MODULES : show all loaded modules
-M : a synonym for -t -D DUMP_MODULES
-t -D DUMP_INCLUDES : show all included configuration files
-t : run syntax check for config files
-T : start without DocumentRoot(s) check
-X : debug mode (only one worker, do not detach)
default:~/Desktop$ apache2
[Sat Dec 07 15:33:06.369259 2024] [core:warn] [pid 1386] AH00111: Config variable ${APACHE_RUN_DIR} is not defined
apache2: Syntax error on line 80 of /etc/apache2/apache2.conf: DefaultRuntimeDir must be a valid directory, absolute or relative to ServerRoot
default:~/Desktop$ source /etc/apache2/envvars
```

查阅相关资料后，发现错误原因是 Apache 服务还没有初始化，使用 `sudo apache2ctl start` 命令可以初始化 Apache 服务对当前服务器的授权。运行后访问调试地址即出现安装成功的界面。



3.4 SYN-FLOOD 攻击

通过 `ifconfig` 命令确认服务端的 IP 地址。

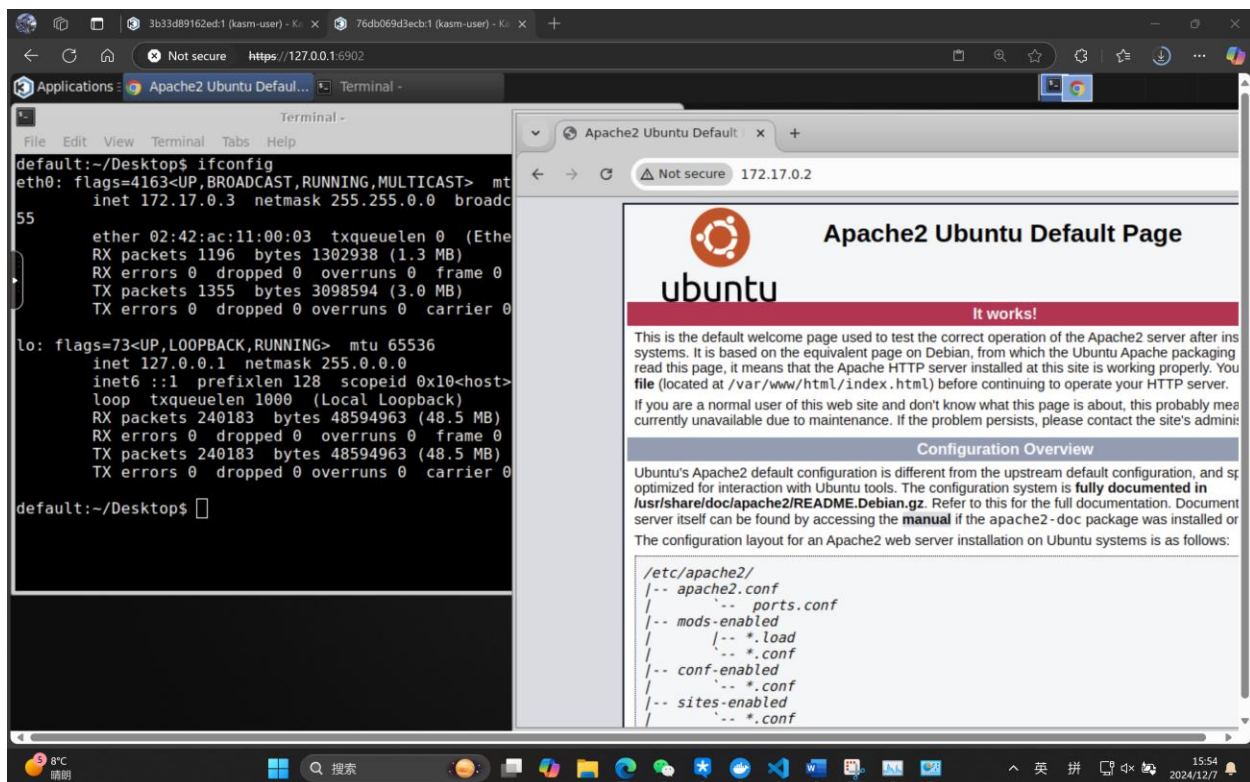
A terminal window titled "Terminal - root@3b33d89162ed: /home/kasm-user/Desktop" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
root@3b33d89162ed:/home/kasm-user/Desktop# apache2ctl start
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
httpd (pid 2149) already running
root@3b33d89162ed:/home/kasm-user/Desktop# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
    ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
    RX packets 43880  bytes 33997301 (33.9 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 34430  bytes 35671614 (35.6 MB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 165180  bytes 33408080 (33.4 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 165180  bytes 33408080 (33.4 MB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

root@3b33d89162ed:/home/kasm-user/Desktop#
```

得到 Server 的 IP 地址为 172.17.0.2，与 Docker 控制台查询到的 IP 地址相同。证明 IP 已经正确配置。在攻击者的浏览器上访问 Server 的 IP 地址，成功出现了安装界面。

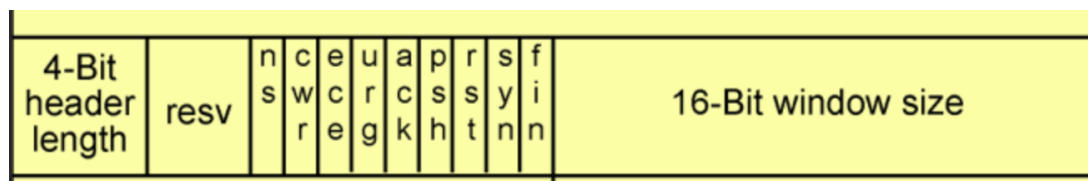


由于在实验一中已经配置好了 Python 环境和 scapy 库，本次无需配置。

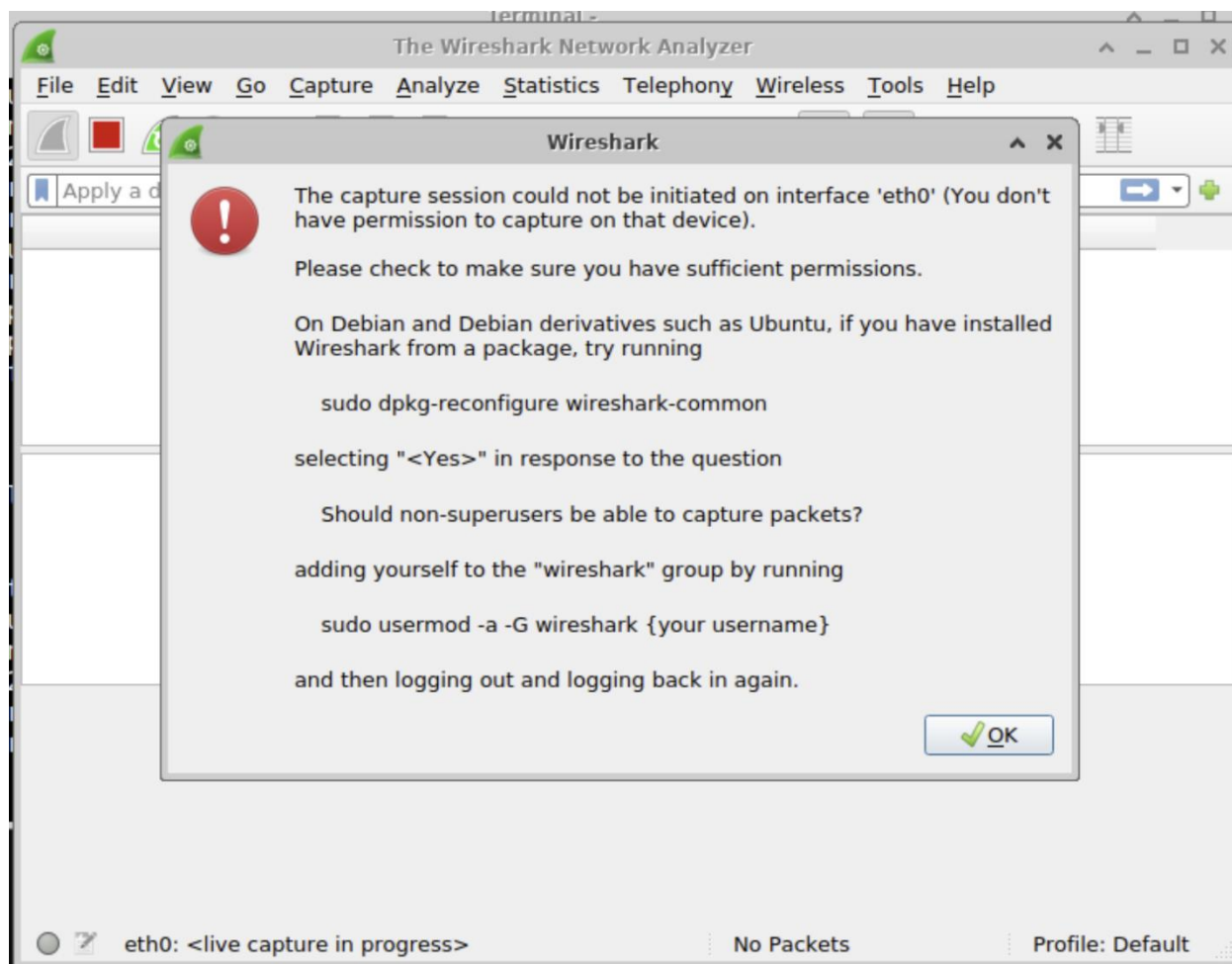
按照实验文档编写程序，编辑器提示无法找到定义的 IP 和 TCP 类。Chayue1 资料发现这是因为在新版本的 scapy 中这两个类被移动到了 `scapy.layers.inet` 模块中。接下来对 `syn_flood` 中 `send` 命令的各个参数作出解释。

- **Send 函数**：从网络层（TCP/IP 体系结构中的第三层）发送指定数据包，本实验中 ‘x’ 参数指明了待发送数据包的类型，启用 ‘loop’ 参数表示不断循环发送数据包，直到程序退出为止。
- **IP 类**：本次实验调用 IP 类的构造函数来创建一个 IPv4 协议数据包报头。其中 ‘src’ 参数填写了报头中的源 IP 地址域，实验中赋值的 `RandIP` 函数即模拟数据包是从各个不同的 IP 地址发送来的。‘dst’ 参数即该 IP 数据包的目的地址，本实验中也即是 Server 在 Docker 局域网网络中的 IP 地址。特别的，由于重载了 ‘/’ 运算符，scapy 允许使用该运算符连接数据包的报头和其负载。在本实验中 IP 类后的内容是该数据包的传输层负载。
- **Fuzz 函数**：该函数可以创建一个符合协议规则且内容随机的数据包。本实验中创建了一个有特定格式的 TCP 数据包。
- **TCP 类**：创建一个 TCP 协议报头，其中 ‘dport’ 参数指明了这个数据包是从 80 号端口发送的。‘flag’ 参数控制了 TCP 的标志位。如图所示，TCP 的标志位由一个 16 位二进制

制数组成，本实验中输入的 0x0002 开启了其中权重第二小的标志位，也就是 SYN 标志。在 TCP 的三次握手机制中，单独 SYN 标志位开启表示请求连接。



在 Server 中运行 Wireshark，发现没有足够的权限进行抓包。



经过查阅资料，发现应将 Wireshark 运行在权限模式下。但是对于本次实验所用的镜像，KasmWeb 并没有设置权限用户（root）的图形接口，即该镜像没有足够的资源在运行 Wireshark 的同时提供抓包权限。

尽管如此，由于 Docker 容器在桥接模式下的网络实现方式是创建虚拟网卡，并且宿主机对容器具有全部的管理权限。因此可以在宿主机中安装 Wireshark 并侦听容器的网卡。这体现了使用 Docker 进行网络实验的灵活性。

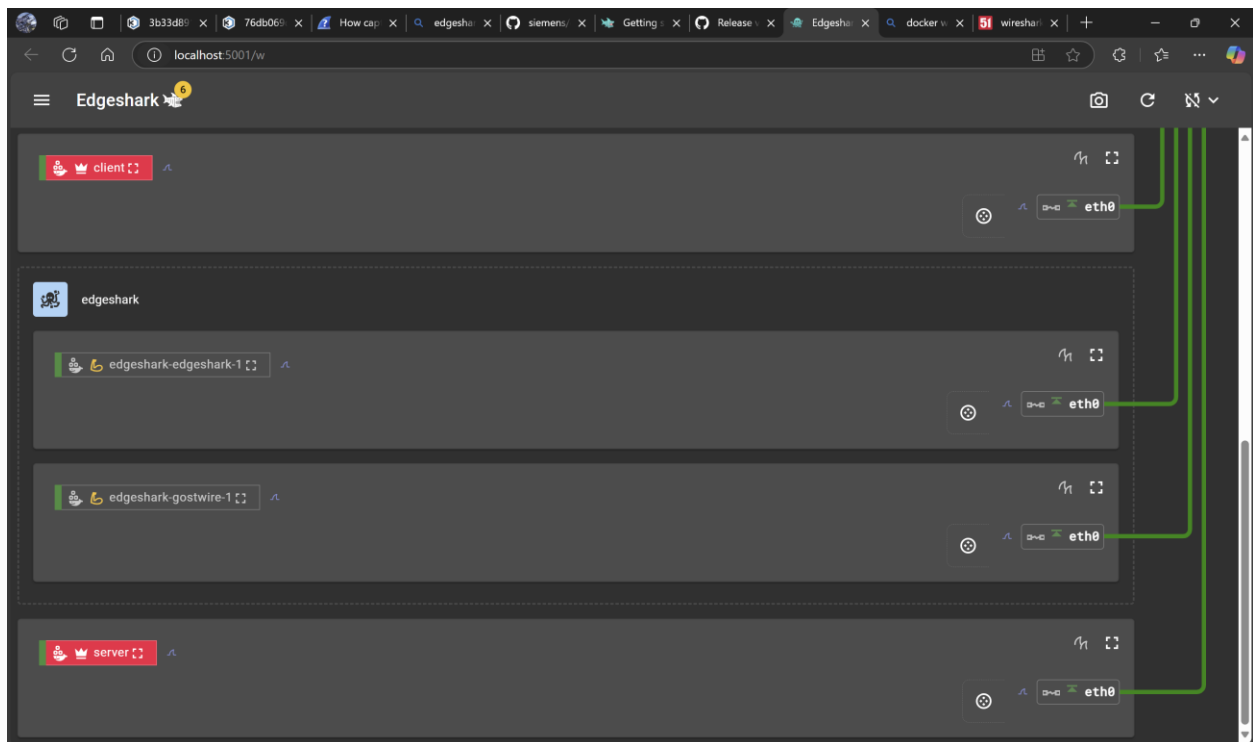
要在 Windows 系统下使用 Wireshark 捕获 Docker 容器中的流量，需要一些额外配置。这是因为 Windows 系统中 Docker 的虚拟网卡创建在 WSL (Hyper-V) 虚拟机中，无法通过 Windows 操作系统直接访问。因此 Windows 环境下的 Wireshark 无法直接连接到容器的网络界面。此时需要用到一个叫做 Edgeshark 的特殊镜像，这个镜像在容器化部署之后会充当 Wireshark 在 WSL 中的探针，把容器网络输入转发到 Windows 系统中的 Wireshark 应用程序中，使得 WSL 中的网络界面对其透明。

首先使用命令行安装 Edgeshark 镜像并启动。

```
ruhaotian@Book3Ultra-Ruhak x + v
}
]
(base) ruhaotian@Book3Ultra-Ruhao:~$ ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
^C
--- 172.17.0.1 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4140ms

(base) ruhaotian@Book3Ultra-Ruhao:~$ docker compose version
Docker Compose version v2.30.3-desktop.1
(base) ruhaotian@Book3Ultra-Ruhao:~$ wget -q --no-cache -O - \
https://github.com/siemens/edgeshark/raw/main/deployments/wget/docker-compose-localhost.yaml \
| DOCKER_DEFAULT_PLATFORM= docker compose -f - up
[+] Running 7/7
  ✓ edgeshark Pulled                                31.0s
  ✓ 4abcf2066143 Pull complete                       6.5s
  ✓ ce715ac60835 Pull complete                       9.4s
  ✓ e27fab22e5de Pull complete                      26.7s
  ✓ gostwire Pulled                                23.6s
  ✓ c6a83fedfae6 Pull complete                      10.7s
  ✓ f22a77d1e784 Pull complete                      19.3s
[+] Running 3/3
  ✓ Network ghost-in-da-edge                        Created      0.0s
  ✓ Container edgeshark-edgeshark-1                 Created      0.1s
  ✓ Container edgeshark-gostwire-1                  Created      0.1s
Attaching to edgeshark-1, gostwire-1
gostwire-1 | INFO[2024-12-07T09:34:35Z] Gostwire "The Sequel" virtual network topology and configuration discovery ser
vice, version 2.6.1
gostwire-1 | INFO[2024-12-07T09:34:35Z] Copyright (c) Siemens AG 2018-2023
gostwire-1 | INFO[2024-12-07T09:34:35Z] available engine process detector plugins: containerd, cri-o, dockerd
```

此时在 Windows 浏览器中访问 `localhost:5001/w` 可见 Edgeshark 已经成功捕获了 Server 和 Client 的网络界面。

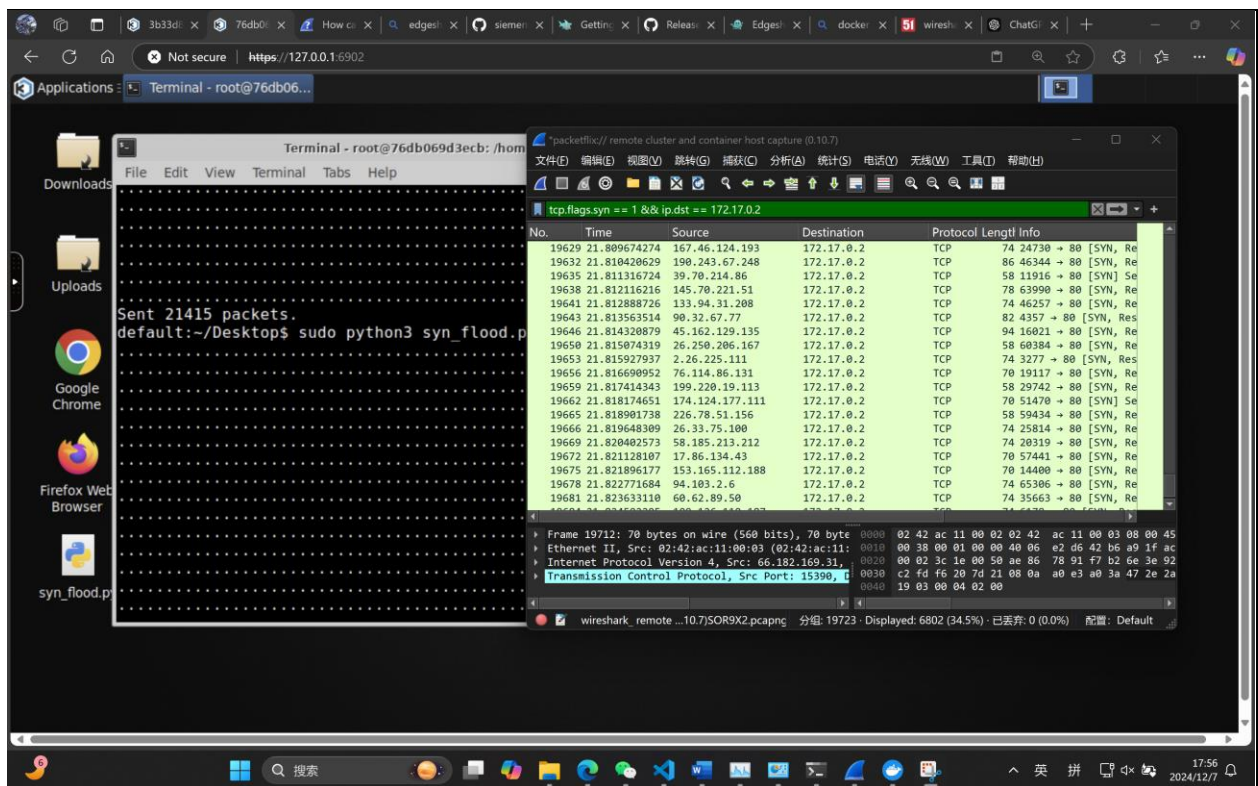


最后，在 Wireshark 中安装 Edgeshark 连接插件，此时 Wireshark 已经完成了抓包准备。

在正式开始抓包之前，最后一个需要考虑的细节是 Wireshark 抓包条件的设置。因为容器和宿主机直接的图形界面也使用网络连接，因此根据攻击程序的发包细节在 Wireshark 的过滤器条件中添加：

```
tcp.flags.syn == 1 && ip.dst == 172.17.0.2
```

开始攻击并启动 Wireshark 抓包，成功捕获到了大量来自不同随机 IP 的数据包。



在 Server 中使用 `netstat -catn` 持续监听，可见每一次刷新都出现了来自随机 IP 的 SYN 请求：

Terminal -

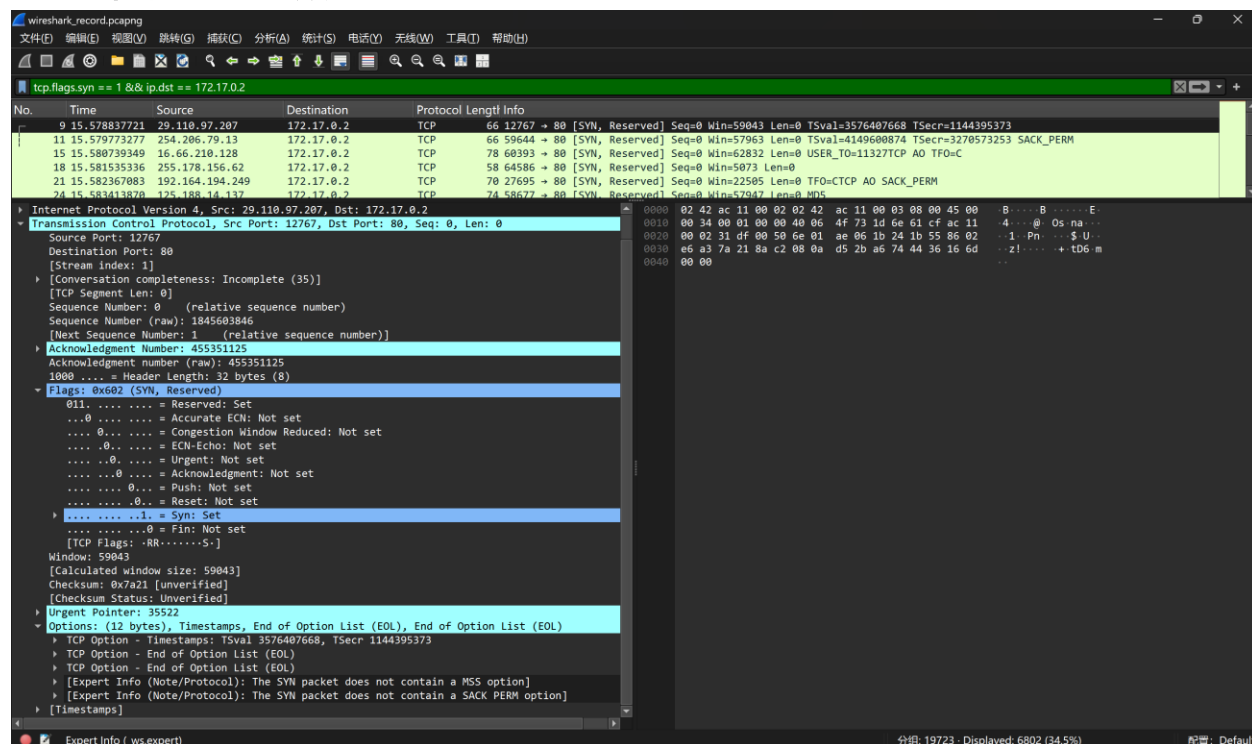
File Edit View Terminal Tabs Help

| Proto | Recv-Q | Send-Q | Local Address | Foreign Address | State |
|-------|--------|--------|-----------------|---------------------|-------------|
| tcp | 0 | 0 | 0.0.0.0:80 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:4904 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:4903 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:4902 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:6901 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 172.17.0.2:80 | 7.113.149.219:60196 | SYN_RECV |
| tcp | 0 | 0 | 172.17.0.2:6901 | 172.17.0.1:45772 | ESTABLISHED |
| tcp | 0 | 0 | 127.0.0.1:37178 | 127.0.0.1:8081 | ESTABLISHED |
| tcp | 0 | 0 | 172.17.0.2:80 | 30.93.187.162:44482 | SYN_RECV |
| tcp6 | 0 | 0 | :::4904 | :::* | LISTEN |
| tcp6 | 0 | 0 | :::4901 | :::* | LISTEN |
| tcp6 | 0 | 0 | :::4903 | :::* | LISTEN |
| tcp6 | 0 | 0 | :::8081 | :::* | LISTEN |
| tcp6 | 0 | 0 | 127.0.0.1:8081 | 127.0.0.1:37178 | ESTABLISHED |

Active Internet connections (servers and established)

| Proto | Recv-Q | Send-Q | Local Address | Foreign Address | State |
|-------|--------|--------|-----------------|-------------------|-------------|
| tcp | 0 | 0 | 0.0.0.0:80 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:4904 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:4903 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:4902 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 0.0.0.0:6901 | 0.0.0.0:* | LISTEN |
| tcp | 0 | 0 | 172.17.0.2:6901 | 172.17.0.1:45772 | ESTABLISHED |
| tcp | 0 | 0 | 127.0.0.1:37178 | 127.0.0.1:8081 | ESTABLISHED |
| tcp | 0 | 0 | 172.17.0.2:80 | 163.48.3.246:3434 | SYN_RECV |
| tcp6 | 0 | 0 | :::4904 | :::* | LISTEN |
| tcp6 | 0 | 0 | :::4901 | :::* | LISTEN |
| tcp6 | 0 | 0 | :::4903 | :::* | LISTEN |
| tcp6 | 0 | 0 | :::8081 | :::* | LISTEN |
| tcp6 | 0 | 0 | 127.0.0.1:8081 | 127.0.0.1:37178 | ESTABLISHED |

3.5 解释 TCP 参数



选取截获的第一个来自 Attacker 的数据包，其 TCP 协议解码信息如上图所示。接下来对其中的每一个字段进行分析。

| 报文字段 | 字段含义 | 具体内容 |
|--|---|--|
| Source Port: 12767 | 源端口 | 发送方的源端口号为 12767 |
| Destination Port: 80 | 目的端口 | 接收方的目的端口为 80 |
| Sequence Number: 0 (relative sequence number) | 序列号（报文第一个字节在字节流中的相对序号） | 相对序列号为 0，表示该报文负载的第一个字节是当前 TCP 连接传输的首个字节 |
| Sequence Number (raw): 1845603846 | 序列号（报文第一个字节在字节流中的序号） | 第一个字节序号为 1845603846，这个起始序号一般随机选择 |
| Acknowledgment number (raw): 0 | 确认号（准备接收的下一个字节的序号，同时表明该序号之前的字节已经全部正确收到） | 该确认号是针对同信道另一个方向数据的，由于这是第一个报文，TCP 双向连接还没有建立，所以确认号无效 |
| 1000 = Header Length: 32 bytes (8) | 偏移量（TCP 报头中可变长度部分的长度，单位是 32 比特） | 该报头中可变长度的选项部分长度为 $8 \times 32\text{bit} = 32\text{B}$ |
| Flags: 0x602 (SYN, Reserved) | TCP 报头中的标志位 | 仅开启了 SYN 标志位（正如攻击程序设置的那样），同时将 Reserved 位置为 011，这是由上文中 Fuzz 函数的随机 |

| | | |
|---|--------------------------|--|
| | | 特性决定的。通过抽样其他 TCP 包，我发这这些包的 Reserved 位都是随机的，验证了这个观点 |
| Window: 59043 | 窗口大小（用于 TCP 滑动窗口协议的流量控制） | 接收方可以接受的字数为 59043 字节，也就是发送方在未收到应答的情况下发送的字节数量不能超过这个值 |
| Checksum: 0x7a21 | 校验和（对报头、伪报头和数据字段进行校验） | 当前报文及其对应的数据字段、伪报头的校验和为 0x7a21 |
| Urgent Pointer: 35522 | 紧急指针所指向的地址 | 由于在 Flag 域中，紧急指针未启用，所以这一栏的值没有实际意义。 |
| TCP Option - Timestamps: TSval 3576407668, TSecr 1144395373 | 选项（可变长度，包含选用的 TCP 选项） | 本报文选用了 Timestamps 选项。该选项由 RFC 1323 引入，用于解决两端往返时延测量和序列号回绕。包含一个发送方时间戳 TSval 和一个回显时间戳 TSecr |

可见相当一部分内容是由攻击程序中的 fuzz 函数随机生成的。

至此，本次实验所有内容结束。

4 心得体会

虽然本次实验难度相比首次实验更低，但是收获同样很大。本次实验解决了上次实验中未能解释的 Docker 网络环境中主机无法 ping 通容器但容器能 ping 通主机的问题——主机的物理网卡对容器可见但容器的虚拟网卡在虚拟机中导致无法通过主机操作系统访问。让我对网卡以及 Docker 网络架构有了进一步的了解。同时我也练习了 Wireshark 抓包技能并复习了计算机网络原理中的 TCP 相关知识。作为计算机系的学生，相信这些知识会对我今后的科研及学习大有帮助。