Lab of Computer Network: NAT **Fall 2024**

Report 11 — Novmeber 20

Lecturer: Wu Qinghua *Completed by: 2022K8009929010 Zhang Jiawei*

11.1 实验内容

1. SNAT
    (a) 运行给定网络拓扑 (nat_topo.py)
    (b) 在 n1, h1, h2, h3 上运行相应脚本
        i. n1: disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh, disable_ipv6.sh
        ii. h1: disable_offloading.sh, disable_ipv6.sh
    (c) 在 n1 上运行 nat 程序：n1# ./nat exp1.conf
    (d) 在 h3 上运行 HTTP 服务：h3# python3 ./http_server.py
    (e) 在 h1, h2 上分别访问 h3 的 HTTP 服务
        i. h1# wget http://159.226.39.123:8000
        ii. h2# wget http://159.226.39.123:8000
2. DNAT
    (a) 运行给定网络拓扑 (nat_topo.py)
    (b) 在 n1, h1, h2, h3 上运行相应脚本
        i. n1: disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh, disable_ipv6.sh
        ii. h1-h3: disable_offloading.sh, disable_ipv6.sh
    (c) 在 n1 上运行 nat 程序：n1# ./nat exp2.conf
    (d) 在 h1, h2 上分别运行 HTTP Server：h1/h2# python3 ./http_server.py
    (e) 在 h3 上分别请求 h1, h2 页面
        i. h3# wget http://159.226.39.43:8000
        ii. h3# wget http://159.226.39.43:8001
3. SDNAT
    (a) 手动构造一个包含两个 nat 的拓扑
        i. h1 <-> n1 <-> n2 <-> h2
        ii. 节点 n1 作为 SNAT, n2 作为 DNAT, 主机 h2 提供 HTTP 服务, 主机 h1 穿过两个 nat 连接到 h2 并获取相应页面

11.2 实验过程

11.2.1 读取配置信息

parse_config 函数用于读取配置文件中的配置信息，先完成 internal 和 external 端口的解析，再查看是否有 DNAT 配置，如果有则解析 DNAT 配置，将其添加到规则列表中。

一个事例配置文件格式如下：

```
internal-iface: n1-eth0
external-iface: n1-eth1

dnat-rules: 159.226.39.23:8002 -> 10.21.0.1:8000
```

我们需要从配置文件读取信息，代码如下：

```
int parse_config(const char *filename)
{
   char *line = (char *)malloc(MAX_LINE_LEN);
   FILE *file = fopen(filename, "r");
   if (!file) {
      fprintf(stderr, "config file do not exist\n");
      free(line);
      return -1;
   }

   while (fgets(line, MAX_LINE_LEN, file)) {
      if (line[0] == 'i') {
         char* internal =line + 16;
         nat.internal_iface = if_name_to_iface(internal);
         continue;
      }
      else if (line[0] == 'e') {
         char* external =line + 16;
         nat.external_iface = if_name_to_iface(external);
         continue;
      }
      else if (line[0] == 'd') {
         struct dnat_rule *rule = (struct dnat_rule *)malloc(sizeof(struct
             dnat_rule));
         memset(rule, 0, sizeof(struct dnat_rule));
         char *drule = line + 12;
         u8 internal_ip[4], external_ip[4];
         sscanf(drule, "%hhu.%hhu.%hhu.%hhu:%hu -> %hhu.%hhu.%hhu.%hhu:%hu",
               &external_ip[3], &external_ip[2], &external_ip[1],
                  &external_ip[0], &rule->external_port,
               &internal_ip[3], &internal_ip[2], &internal_ip[1],
                  &internal_ip[0], &rule->internal_port);
         rule->external_ip = external_ip[3] << 24 | external_ip[2] << 16 |
            external_ip[1] << 8 | external_ip[0];
         rule->internal_ip = internal_ip[3] << 24 | internal_ip[2] << 16 |
            internal_ip[1] << 8 | internal_ip[0];
         init_list_head(&rule->list);
         list_add_tail(&rule->list, &nat.rules);
         nat.assigned_ports[rule->external_port] = 1;
         continue;
      }
   }
   fclose(file);
   free(line);
   return 0;
}
```

11.2.2 网络地址转换

在进行地址转换前，我们需要区分数据包的发送方向。当源地址为内部地址，且目的地址为外部地址时，方向为 DIR_OUT；当源地址为外部地址，且目的地址为 external_iface 地址时，方向为 DIR_IN；否则为 DIR_INVALID：

```
// determine the direction of the packet, DIR_IN / DIR_OUT / DIR_INVALID
static int get_packet_direction(char *packet)
{
   // fprintf(stdout, "TODO: determine the direction of this packet.\n");
   struct iphdr *ip = packet_to_ip_hdr(packet);
   u32 saddr = ntohl(ip->saddr);
   u32 daddr = ntohl(ip->daddr);
   rt_entry_t *src = longest_prefix_match(saddr);
   rt_entry_t *dst = longest_prefix_match(daddr);

   if (src->iface == nat.internal_iface && dst->iface == nat.external_iface)
      return DIR_OUT;
   else if (src->iface == nat.external_iface && daddr == nat.external_iface->ip)
      return DIR_IN;
   else
      return DIR_INVALID;
}
```

根据数据包的方向类型，先丢弃掉不可达的数据包和非 TCP 协议的数据包，再根据数据包的类型进行地址翻译：

```
void nat_translate_packet(iface_info_t *iface, char *packet, int len)
{
   int dir = get_packet_direction(packet);
   if (dir == DIR_INVALID) {
      log(ERROR, "invalid packet direction, drop it.");
      icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_HOST_UNREACH);
      free(packet);
      return ;
   }

   struct iphdr *ip = packet_to_ip_hdr(packet);
   if (ip->protocol != IPPROTO_TCP) {
      log(ERROR, "received non-TCP packet (0x%0hhx), drop it", ip->protocol);
      free(packet);
      return ;
   }

   do_translation(iface, packet, len, dir);
}
```

数据包的地址翻译是本次试验的重点，分为 TCP 已连接和未连接两种情况。首先，我们需要根据数据包的方向来确定远端地址和端口，并计算出相应哈希值以查找连接或创建新连接。查找连接时，需要匹配的条件为：远端地址、远端端口、本地地址、本地端口（如果是 SNAT 则为外部地址和端口，如果是 DNAT 则为内部地址和端口）。若匹配到则修改头部中的源地址和目的地址，也要根据数据包头部修改映射表中相应内容，然后转发即可。若未匹配到，先检查该数据包是否是 SYN 包，若不是则直接丢弃，回复地址不可达；若是 SYN 包，则创建新连接，根据数据包的方向类型，若为公网访问内网，直接在已有映射规则中查找，查找到时则修改头部中的源地址和目的地址，也要根据数据包头部修改映射表中相应内容，然后转发即可；若为内网访问公网，则遍历找到一个未使用的端口，创建新的映射规则，修改头部中的源地址和目的地址，也要根据数据包头部修改映射表中相应内容，然后转发即可。最后，如果以上过程结束之后仍未知道如何处理该数据包，则直接丢弃，回复地址不可达：

```
// do translation for the packet: replace the ip/port, recalculate ip & tcp
// checksum, update the statistics of the tcp connection
void do_translation(iface_info_t *iface, char *packet, int len, int dir)
{
   // fprintf(stdout, "TODO: do translation for this packet.\n");
   struct iphdr *iphdr = packet_to_ip_hdr(packet);
   struct tcphdr *tcphdr = packet_to_tcp_hdr(packet);
   u32 daddr = ntohl(iphdr->daddr);
   u32 saddr = ntohl(iphdr->saddr);
   u32 raddr = (dir == DIR_IN) ? saddr : daddr;
   u16 sport = ntohs(tcphdr->sport);
   u16 dport = ntohs(tcphdr->dport);
   u16 rport = (dir == DIR_IN) ? sport : dport;

   char *str = (char *)malloc(6);
   memset(str, 0, 6);
   memcpy(str, &raddr, 4);
   memcpy(str + 4, &rport, 2);
   u8 hash = hash8(str, 6);
   free(str);
   struct list_head *head = &nat.nat_mapping_list[hash];
   struct nat_mapping *entry = NULL;

   pthread_mutex_lock(&nat.lock);
   list_for_each_entry(entry, head, list) {
      if (raddr != entry->remote_ip || rport != entry->remote_port)
         continue;

      int clear = (tcphdr->flags & TCP_RST) ? 1 : 0;

      if (dir == DIR_IN) {
         if (daddr != entry->external_ip || dport != entry->external_port)
            continue;

         iphdr->daddr = htonl(entry->internal_ip);
         tcphdr->dport = htons(entry->internal_port);
         entry->conn.external_fin = (tcphdr->flags & TCP_FIN) ? 1 : 0;
         entry->conn.external_seq_end = tcp_seq_end(iphdr, tcphdr);
         if (tcphdr->flags & TCP_ACK)
            entry->conn.external_ack = tcphdr->ack;
      }
      else {
         if (saddr != entry->internal_ip || sport != entry->internal_port)
            continue;

         iphdr->saddr = htonl(entry->external_ip);
         tcphdr->sport = htons(entry->external_port);
         entry->conn.internal_fin = (tcphdr->flags & TCP_FIN) ? 1 : 0;
         entry->conn.internal_seq_end = tcp_seq_end(iphdr, tcphdr);
         if (tcphdr->flags & TCP_ACK)
            entry->conn.internal_ack = tcphdr->ack;
      }
      pthread_mutex_unlock(&nat.lock);

      entry->update_time = time(NULL);
      tcphdr->checksum = tcp_checksum(iphdr, tcphdr);
      iphdr->checksum = ip_checksum(iphdr);
      ip_send_packet(packet, len);

      if (clear) {
         nat.assigned_ports[entry->external_port] = 0;
         list_delete_entry(&(entry->list));
         free(entry);
      }
      return;
   }

   if ((tcphdr->flags & TCP_SYN) == 0) {
      fprintf(stderr, "Invalid packet!\n");
      icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_HOST_UNREACH);
      free(packet);
      pthread_mutex_unlock(&nat.lock);
      return;
   }

   if (dir == DIR_IN) {
      struct dnat_rule *rule;
      list_for_each_entry(rule, &nat.rules, list) {
         if (daddr == rule->external_ip && dport == rule->external_port) {
            struct nat_mapping *new = (struct nat_mapping *) malloc
                (sizeof(struct nat_mapping));
            list_add_tail(&new->list, head);

            new->remote_ip = raddr;
            new->remote_port = rport;
            new->external_ip = rule->external_ip;
            new->external_port = rule->external_port;
            new->internal_ip = rule->internal_ip;
            new->internal_port = rule->internal_port;
            new->conn.external_fin = ((tcphdr->flags & TCP_FIN) != 0);
            new->conn.external_seq_end = tcp_seq_end(iphdr, tcphdr);
            if (tcphdr->flags & TCP_ACK)
               new->conn.external_ack = tcphdr->ack;
            new->update_time = time(NULL);
            pthread_mutex_unlock(&nat.lock);

            iphdr->daddr = htonl(rule->internal_ip);
            tcphdr->dport = htons(rule->internal_port);
            tcphdr->checksum = tcp_checksum(iphdr, tcphdr);
            iphdr->checksum = ip_checksum(iphdr);
            ip_send_packet(packet, len);
            return;
         }
      }
   }
   else {
      u16 pid;
      for (pid = NAT_PORT_MIN; pid <= NAT_PORT_MAX; ++pid) {
         if (!nat.assigned_ports[pid]) {
            struct nat_mapping *new = (struct nat_mapping *) malloc(sizeof(struct
                nat_mapping));
            list_add_tail(&new->list, head);

            new->remote_ip = raddr;
            new->remote_port = rport;
            new->external_ip = nat.external_iface->ip;
            new->external_port = pid;
            new->internal_ip = saddr;
            new->internal_port = sport;
            new->conn.internal_fin = (tcphdr->flags & TCP_FIN) != 0;
            new->conn.internal_seq_end = tcp_seq_end(iphdr, tcphdr);
            if (tcphdr->flags & TCP_ACK)
               new->conn.internal_ack = tcphdr->ack;
            new->update_time = time(NULL);
            pthread_mutex_unlock(&nat.lock);

            iphdr->saddr = htonl(new->external_ip);
            tcphdr->sport = htons(new->external_port);
            tcphdr->checksum = tcp_checksum(iphdr, tcphdr);
            iphdr->checksum = ip_checksum(iphdr);
            ip_send_packet(packet, len);
            return;
         }
      }
   }

   icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_HOST_UNREACH);
```

```
    free(packet);
}
```

11.2.3 映射表的老化

老化线程每秒唤醒一次，删除超时未传输和已经握手完毕断开连接的映射规则，释放端口：

```
void *nat_timeout()
{
    while (1) {
        // fprintf(stdout, "TODO: sweep finished flows periodically.\n");
        sleep(1);
        pthread_mutex_lock(&nat.lock);
        for (int i = 0; i < HASH_8BITS; i++) {
            struct nat_mapping *entry = NULL, *map_q = NULL;
            list_for_each_entry_safe(entry, map_q, &nat.nat_mapping_list[i], list) {
                if (time(NULL) - entry->update_time > TCP_ESTABLISHED_TIMEOUT ||
                    is_flow_finished(&entry->conn)) {
                    nat.assigned_ports[entry->external_port] = 0;
                    list_delete_entry(&entry->list);
                    free(entry);
                }
            }
        }
        pthread_mutex_unlock(&nat.lock);
    }

    return NULL;
}
```

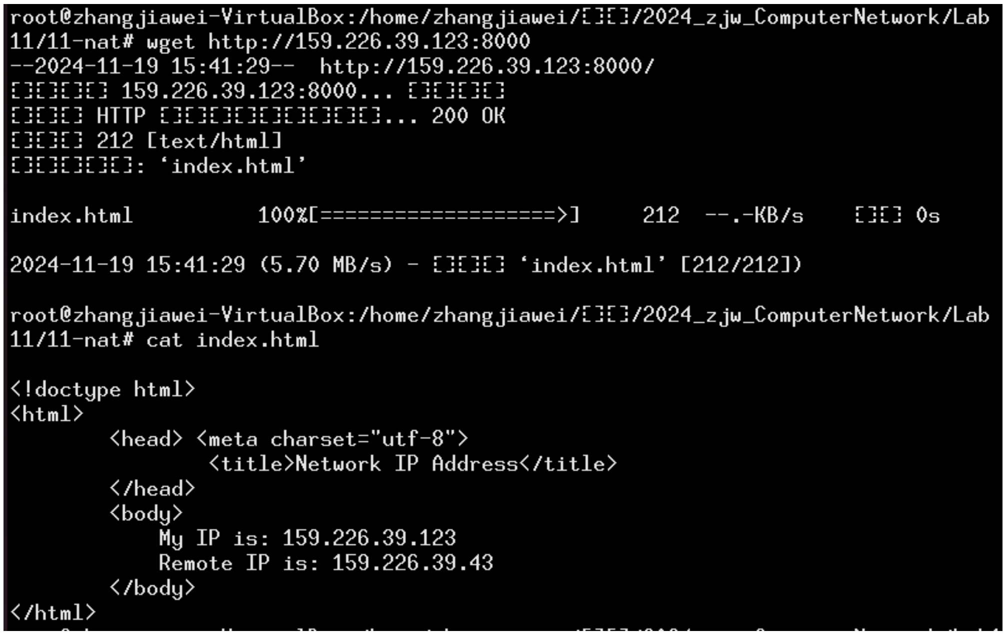
11.3 实验结果

11.3.1 SNAT

在 n1 上运行 nat 程序，读入配置文件 exp1.conf，配置文件如下：

```
internal-iface: n1-eth0
external-iface: n1-eth1
```

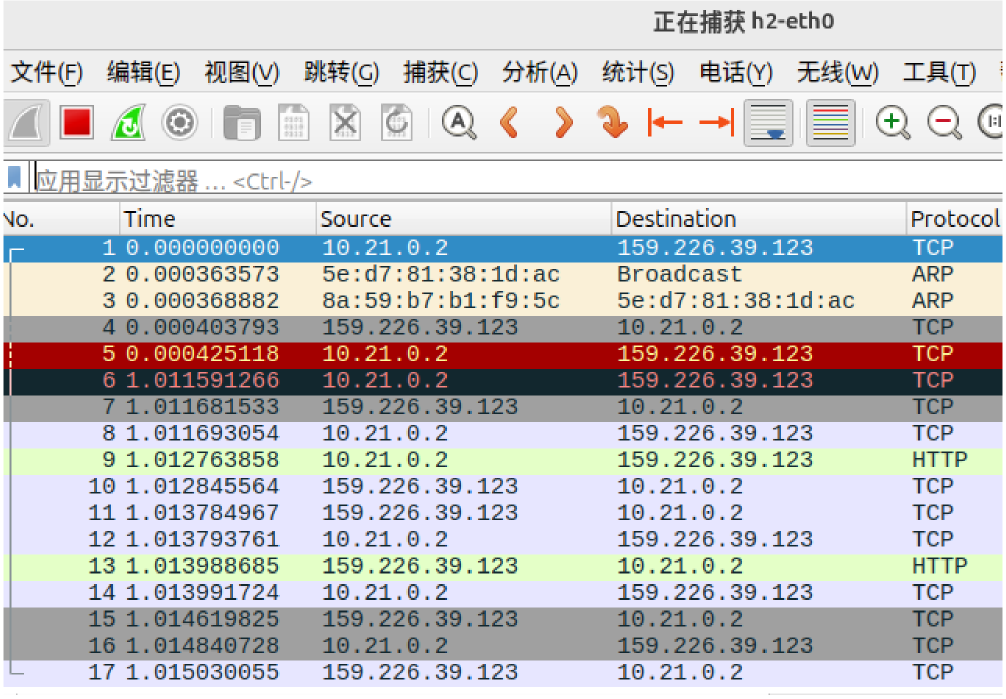
在 h2 上进行 wget 操作，结果如下：



```
root@zhangjiawei-VirtualBox:/home/zhangjiawei/□□/2024_zjw_ComputerNetwork/Lab
11/11-nat# wget http://159.226.39.123:8000
--2024-11-19 15:41:29--  http://159.226.39.123:8000/
□□□□ 159.226.39.123:8000... □□□□
□□□ HTTP □□□□□□□□□... 200 OK
□□□ 212 [text/html]
□□□□□: ‘index.html’

index.html          100%[===================>]     212  --.-KB/s    □□ 0s

2024-11-19 15:41:29 (5.70 MB/s) - □□□ ‘index.html’ [212/212])

root@zhangjiawei-VirtualBox:/home/zhangjiawei/□□/2024_zjw_ComputerNetwork/Lab
11/11-nat# cat index.html

<!doctype html>
<html>
        <head> <meta charset="utf-8">
                <title>Network IP Address</title>
        </head>
        <body>
            My IP is: 159.226.39.123
            Remote IP is: 159.226.39.43
        </body>
</html>
```

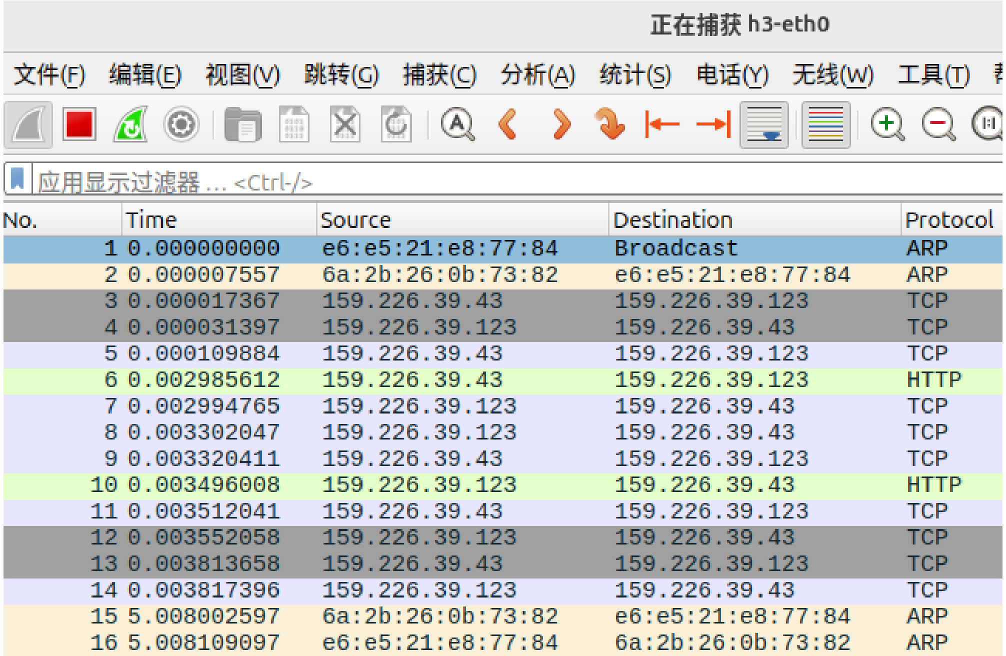
图 **11.1.** h2 获取 h3 页面

使用 wireshark 抓包，结果如下：


正在捕获 h2-eth0
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T)
应用显示过滤器 … <Ctrl-/>

No.	Time	Source	Destination	Protocol
1	0.000000000	10.21.0.2	159.226.39.123	TCP
2	0.000363573	5e:d7:81:38:1d:ac	Broadcast	ARP
3	0.000368882	8a:59:b7:b1:f9:5c	5e:d7:81:38:1d:ac	ARP
4	0.000403793	159.226.39.123	10.21.0.2	TCP
5	0.000425118	10.21.0.2	159.226.39.123	TCP
6	1.011591266	10.21.0.2	159.226.39.123	TCP
7	1.011681533	159.226.39.123	10.21.0.2	TCP
8	1.011693054	10.21.0.2	159.226.39.123	TCP
9	1.012763858	10.21.0.2	159.226.39.123	HTTP
10	1.012845564	159.226.39.123	10.21.0.2	TCP
11	1.013784967	159.226.39.123	10.21.0.2	TCP
12	1.013793761	10.21.0.2	159.226.39.123	TCP
13	1.013988685	159.226.39.123	10.21.0.2	HTTP
14	1.013991724	10.21.0.2	159.226.39.123	TCP
15	1.014619825	159.226.39.123	10.21.0.2	TCP
16	1.014840728	10.21.0.2	159.226.39.123	TCP
17	1.015030055	159.226.39.123	10.21.0.2	TCP

图 **11.2.** h2 获取 h3 页面抓包 h2 结果


正在捕获 h3-eth0
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T)
应用显示过滤器 … <Ctrl-/>

No.	Time	Source	Destination	Protocol
1	0.000000000	e6:e5:21:e8:77:84	Broadcast	ARP
2	0.000007557	6a:2b:26:0b:73:82	e6:e5:21:e8:77:84	ARP
3	0.000017367	159.226.39.43	159.226.39.123	TCP
4	0.000031397	159.226.39.123	159.226.39.43	TCP
5	0.000109884	159.226.39.43	159.226.39.123	TCP
6	0.002985612	159.226.39.43	159.226.39.123	HTTP
7	0.002994765	159.226.39.123	159.226.39.43	TCP
8	0.003302047	159.226.39.123	159.226.39.43	TCP
9	0.003320411	159.226.39.43	159.226.39.123	TCP
10	0.003496008	159.226.39.123	159.226.39.43	HTTP
11	0.003512041	159.226.39.43	159.226.39.123	TCP
12	0.003552058	159.226.39.123	159.226.39.43	TCP
13	0.003813658	159.226.39.43	159.226.39.123	TCP
14	0.003817396	159.226.39.123	159.226.39.43	TCP
15	5.008002597	6a:2b:26:0b:73:82	e6:e5:21:e8:77:84	ARP
16	5.008109097	e6:e5:21:e8:77:84	6a:2b:26:0b:73:82	ARP

图 **11.3.** h2 获取 h3 页面抓包 h3 结果

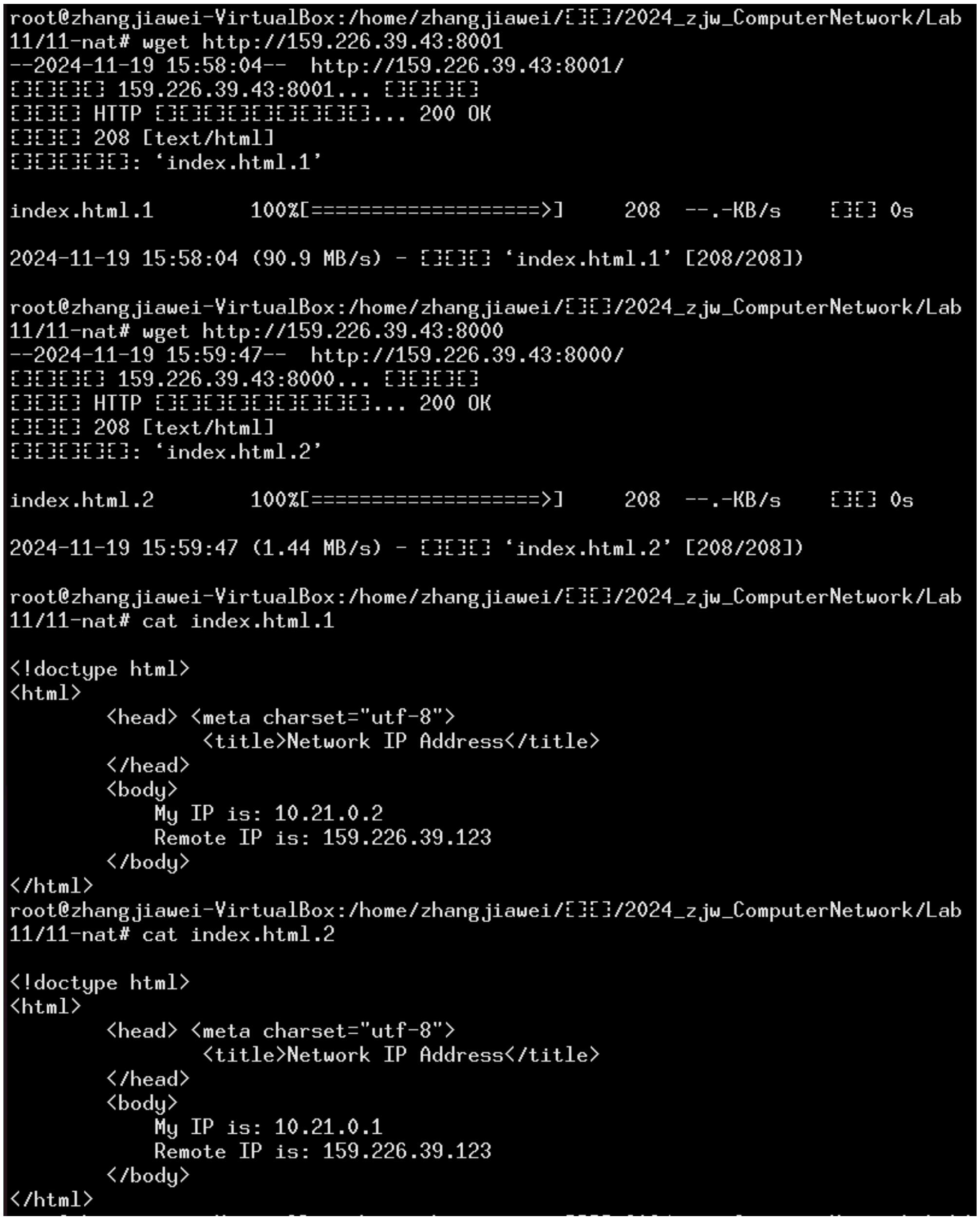
可见，h2 成功获取到了 h3 的页面，且 NAT 成功将私有地址转换为公有地址。

11.3.2 DNAT

在 n1 上运行 nat 程序，读入配置文件 exp2.conf，配置文件如下：

```
internal-iface: n1-eth0
external-iface: n1-eth1

dnat-rules: 159.226.39.43:8000 -> 10.21.0.1:8000
dnat-rules: 159.226.39.43:8001 -> 10.21.0.2:8000
```

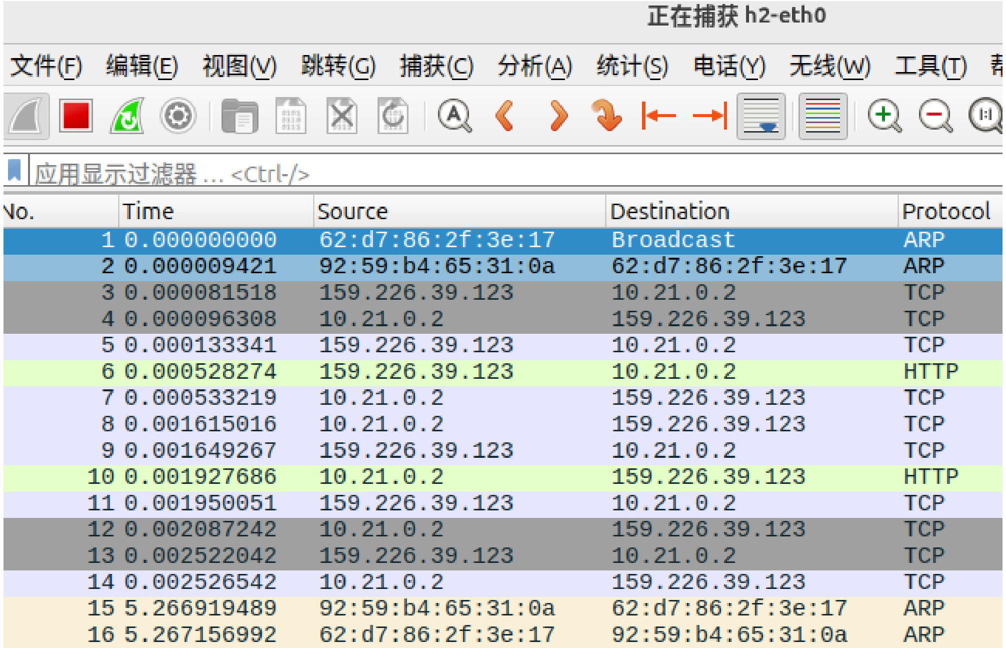
在 h3 上进行 wget 操作，结果如下：



```
root@zhangjiawei-VirtualBox:/home/zhangjiawei/□□/2024_zjw_ComputerNetwork/Lab
11/11-nat# wget http://159.226.39.43:8001
--2024-11-19 15:58:04--  http://159.226.39.43:8001/
□□□□ 159.226.39.43:8001... □□□□
□□□ HTTP □□□□□□□□□... 200 OK
□□□ 208 [text/html]
□□□□□: ‘index.html.1’

index.html.1        100%[===================>]     208  --.-KB/s    □□ 0s

2024-11-19 15:58:04 (90.9 MB/s) - □□□ ‘index.html.1’ [208/208])

root@zhangjiawei-VirtualBox:/home/zhangjiawei/□□/2024_zjw_ComputerNetwork/Lab
11/11-nat# wget http://159.226.39.43:8000
--2024-11-19 15:59:47--  http://159.226.39.43:8000/
□□□□ 159.226.39.43:8000... □□□□
□□□ HTTP □□□□□□□□□... 200 OK
□□□ 208 [text/html]
□□□□□: ‘index.html.2’

index.html.2        100%[===================>]     208  --.-KB/s    □□ 0s

2024-11-19 15:59:47 (1.44 MB/s) - □□□ ‘index.html.2’ [208/208])

root@zhangjiawei-VirtualBox:/home/zhangjiawei/□□/2024_zjw_ComputerNetwork/Lab
11/11-nat# cat index.html.1

<!doctype html>
<html>
        <head> <meta charset="utf-8">
                <title>Network IP Address</title>
        </head>
        <body>
            My IP is: 10.21.0.2
            Remote IP is: 159.226.39.123
        </body>
</html>
root@zhangjiawei-VirtualBox:/home/zhangjiawei/□□/2024_zjw_ComputerNetwork/Lab
11/11-nat# cat index.html.2

<!doctype html>
<html>
        <head> <meta charset="utf-8">
                <title>Network IP Address</title>
        </head>
        <body>
            My IP is: 10.21.0.1
            Remote IP is: 159.226.39.123
        </body>
</html>
```

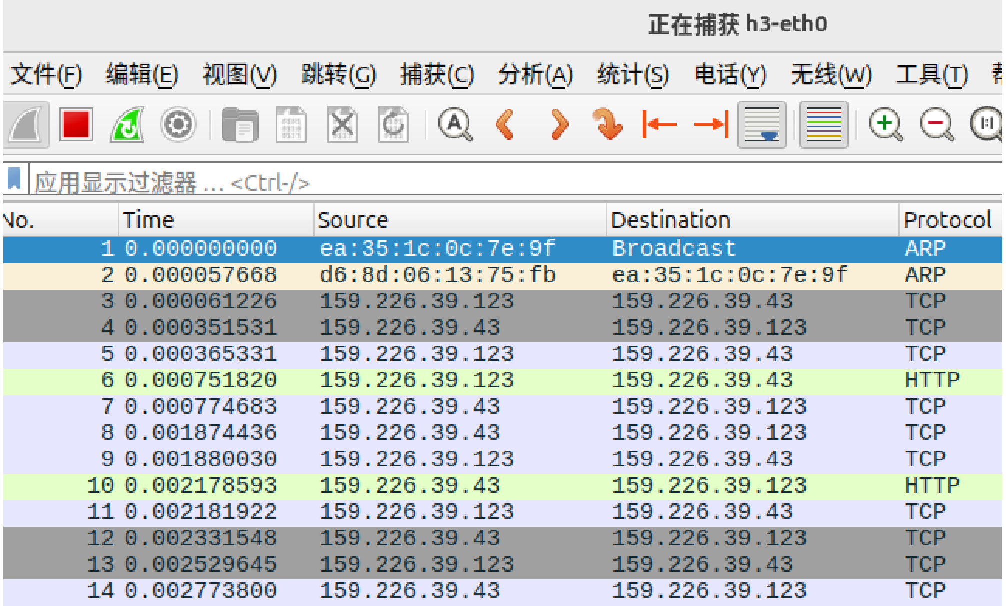
图 **11.4.** h3 获取 h1, h2 页面

使用 wireshark 抓包，结果如下：


正在捕获 h2-eth0
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮
应用显示过滤器 … <Ctrl-/>

No.	Time	Source	Destination	Protocol
1	0.000000000	62:d7:86:2f:3e:17	Broadcast	ARP
2	0.000009421	92:59:b4:65:31:0a	62:d7:86:2f:3e:17	ARP
3	0.000081518	159.226.39.123	10.21.0.2	TCP
4	0.000096308	10.21.0.2	159.226.39.123	TCP
5	0.000133341	159.226.39.123	10.21.0.2	TCP
6	0.000528274	159.226.39.123	10.21.0.2	HTTP
7	0.000533219	10.21.0.2	159.226.39.123	TCP
8	0.001615016	10.21.0.2	159.226.39.123	TCP
9	0.001649267	159.226.39.123	10.21.0.2	TCP
10	0.001927686	10.21.0.2	159.226.39.123	HTTP
11	0.001950051	159.226.39.123	10.21.0.2	TCP
12	0.002087242	10.21.0.2	159.226.39.123	TCP
13	0.002522042	159.226.39.123	10.21.0.2	TCP
14	0.002526542	10.21.0.2	159.226.39.123	TCP
15	5.266919489	92:59:b4:65:31:0a	62:d7:86:2f:3e:17	ARP
16	5.267156992	62:d7:86:2f:3e:17	92:59:b4:65:31:0a	ARP

图 **11.5.** h3 获取 h2 页面抓包 h2 结果


正在捕获 h3-eth0
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮
应用显示过滤器 … <Ctrl-/>

No.	Time	Source	Destination	Protocol
1	0.000000000	ea:35:1c:0c:7e:9f	Broadcast	ARP
2	0.000057668	d6:8d:06:13:75:fb	ea:35:1c:0c:7e:9f	ARP
3	0.000061226	159.226.39.123	159.226.39.43	TCP
4	0.000351531	159.226.39.43	159.226.39.123	TCP
5	0.000365331	159.226.39.123	159.226.39.43	TCP
6	0.000751820	159.226.39.123	159.226.39.43	HTTP
7	0.000774683	159.226.39.43	159.226.39.123	TCP
8	0.001874436	159.226.39.43	159.226.39.123	TCP
9	0.001880030	159.226.39.123	159.226.39.43	TCP
10	0.002178593	159.226.39.43	159.226.39.123	HTTP
11	0.002181922	159.226.39.123	159.226.39.43	TCP
12	0.002331548	159.226.39.43	159.226.39.123	TCP
13	0.002529645	159.226.39.123	159.226.39.43	TCP
14	0.002773800	159.226.39.43	159.226.39.123	TCP

图 **11.6.** h3 获取 h2 页面抓包 h3 结果

可见，h3 成功获取到了 h1 和 h2 的页面，且 NAT 成功将公有地址转换为私有地址。

11.3.3 SDNAT

手动构造一个包含两个 nat 的拓扑如下：

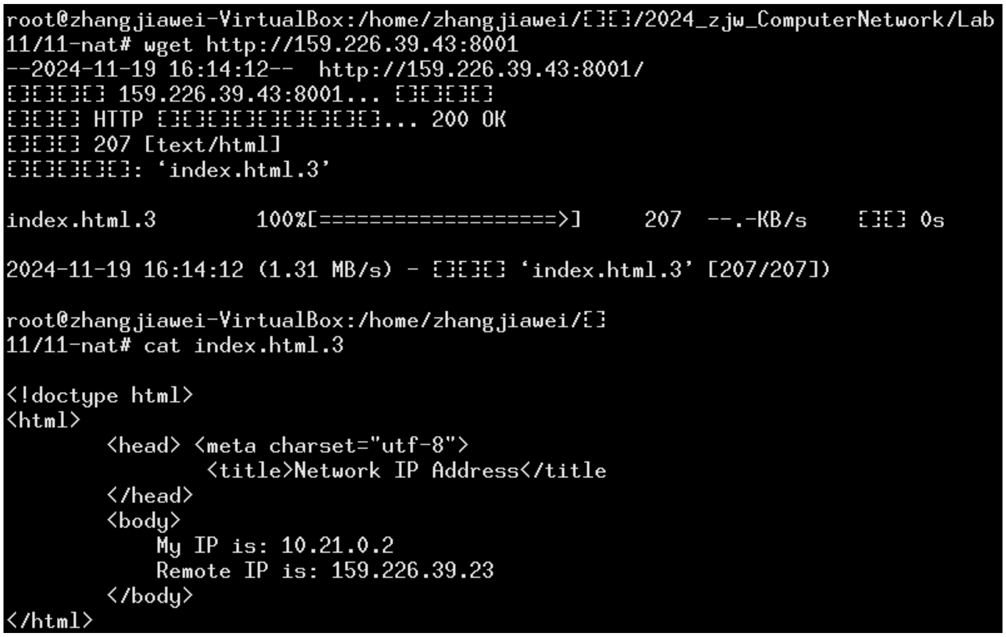
```
h1, h2, n1, n2 = net.get('h1', 'h2', 'n1', 'n2')

h1.cmd('ifconfig h1-eth0 10.21.0.1/16')
h1.cmd('route add default gw 10.21.0.254')

h2.cmd('ifconfig h2-eth0 10.21.0.2/16')
h2.cmd('route add default gw 10.21.0.254')

n1.cmd('ifconfig n1-eth0 10.21.0.254/16')
n1.cmd('ifconfig n1-eth1 159.226.39.23/24')

n2.cmd('ifconfig n2-eth0 10.21.0.254/16')
n2.cmd('ifconfig n2-eth1 159.226.39.43/24')
```

编写配置文件，分别在 n1 和 n2 上运行 nat 程序并读入配置文件：

```
internal-iface: n1-eth0
external-iface: n1-eth1

dnat-rules: 159.226.39.23:8002 -> 10.21.0.1:8000
```

```
internal-iface: n2-eth0
external-iface: n2-eth1

dnat-rules: 159.226.39.43:8001 -> 10.21.0.2:8000
```

令 h2 提供 http 服务，在 h1 上进行 wget 操作，结果如下：



```
root@zhangjiawei-VirtualBox:/home/zhangjiawei/□□/2024_zjw_ComputerNetwork/Lab
11/11-nat# wget http://159.226.39.43:8001
--2024-11-19 16:14:12--  http://159.226.39.43:8001/
□□□□ 159.226.39.43:8001... □□□□
□□□ HTTP □□□□□□□□□□... 200 OK
□□□ 207 [text/html]
□□□□□□: 'index.html.3'

index.html.3        100%[===================>]     207  --.-KB/s    □□ 0s

2024-11-19 16:14:12 (1.31 MB/s) - □□□ 'index.html.3' [207/207])

root@zhangjiawei-VirtualBox:/home/zhangjiawei/□
11/11-nat# cat index.html.3

<!doctype html>
<html>
        <head> <meta charset="utf-8">
                <title>Network IP Address</title
        </head>
        <body>
            My IP is: 10.21.0.2
            Remote IP is: 159.226.39.23
        </body>
</html>
```

图 **11.7.** h1 获取 h2 页面

可见,h1 成功获取到了 h2 的页面,且 NAT 转换成功。

11.4 实验总结

本次实验主要学习了 NAT 的实现原理, 通过实验了解了 SNAT、DNAT 和 SDNAT 的实现方法, 掌握了 NAT 的实现过程,与理论知识相结合,加深了对 NAT 的理解。