实验 10 • 网络地址转换实验

吴嘉皓
2015K8009915007

一、实验内容

修改 nat.c 文件，实现以下功能：

1. NAT 映射表管理：
   1) 维护 NAT 连接映射表，支持映射的**添加、查找、更新**和**老化**操作；
2. 数据包的转换操作
   1) 对到达的合法数据包，进行 IP 和 Port **转换**操作，**更新**头部字段，并**转发**数据包；
   2) 对于到达的非法数据包，回复 ICMP Destination Host Unreachable

二、实验流程

（一）代码目录

```
2015K8009915007_吴嘉皓_10.tar.gz
├── Makefile
├── Makefile.bak
├── include
│   ├── arp.h
│   ├── arpcache.h
│   ├── base.h
│   ├── checksum.h
│   ├── ether.h
│   ├── hash.h
│   ├── icmp.h
│   ├── ip.h
│   ├── list.h
│   ├── log.h
│   ├── nat.h
│   ├── packet.h
│   ├── rtable.h
│   ├── tcp.h
│   └── types.h
├── libipstack.a
├── main.c
└── nat.c
```

```
├── nat_topo.py
├── scripts
│   ├── disable_arp.sh
│   ├── disable_icmp.sh
│   ├── disable_ip_forward.sh
│   ├── disable_offloading.sh
│   └── disable_tcp_rst.sh
├── test.c
└── 实验 10-网络地址转换-实验报告.pdf
```

（二）实验流程

1. 在 10-nat 目录下输入如下命令：

```
make
sudo python nat_topo.py
mininet> xterm n1 h1 h2
n1> ./nat
h2> python -m SimpleHTTPServer
h1> wget http://159.226.39.123:8000
```

三、实验结果

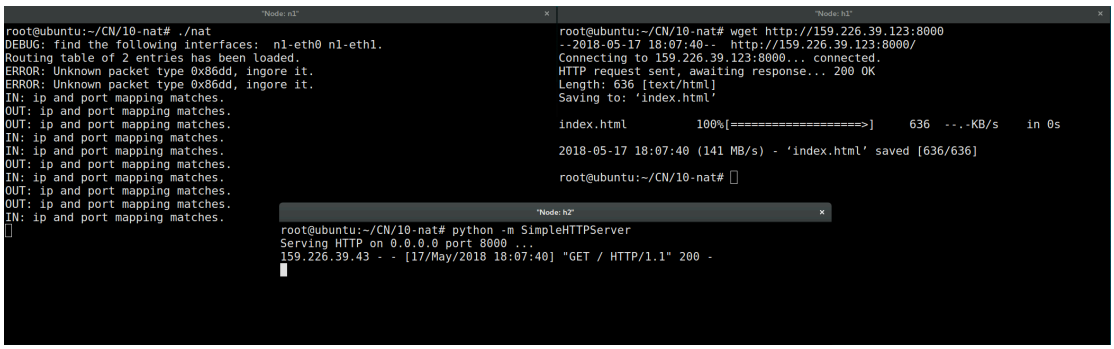

图 1 获取网页结果图

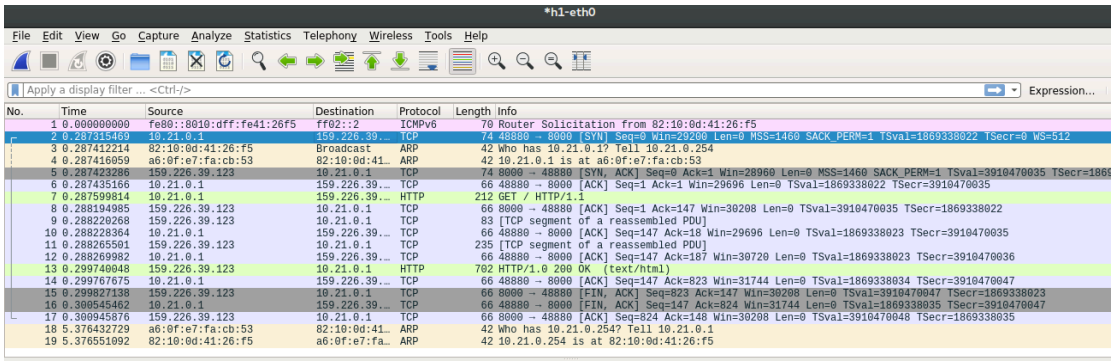

图 2 h1 节点的抓包结果

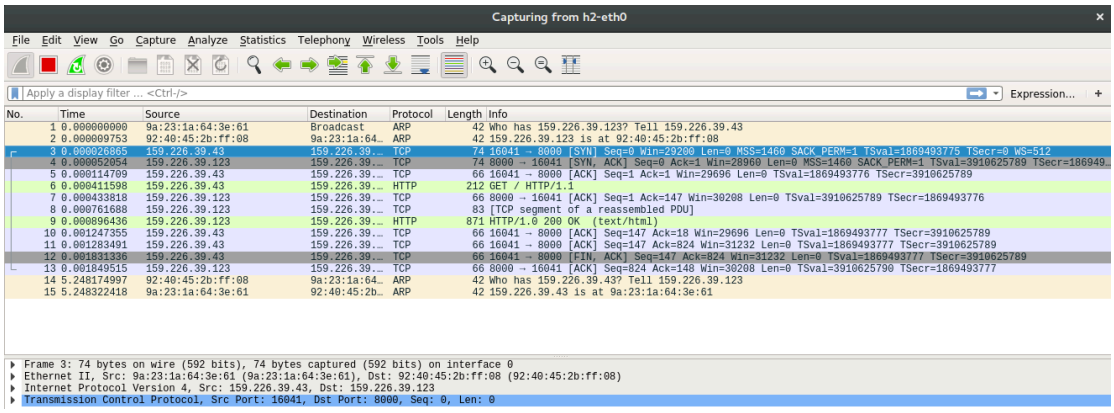

图 3 h2 节点的抓包结果

四、结果分析

（一）结果分析

由图 1 可知，h1 成功将 h2 上生成的 html 网页获取到了本机 h1 上。

通过 wireshark 抓包，我们得到了图 2 和图 3 的结果，并且我们能够从这一次抓包中看出 h1 获取 h2 上网页的整个过程。

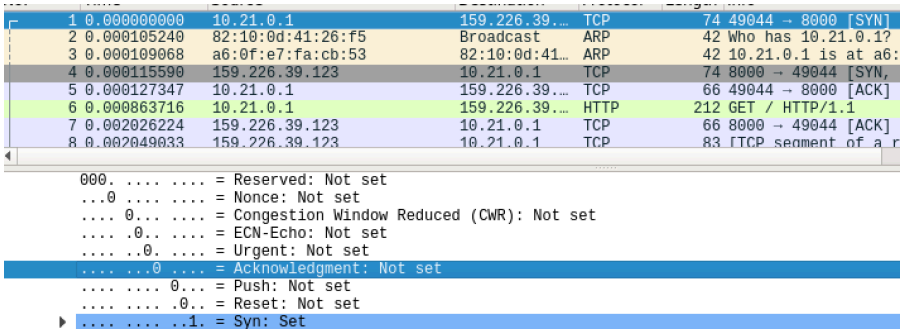

图 5 h2 向 h1 发送建立连接信号

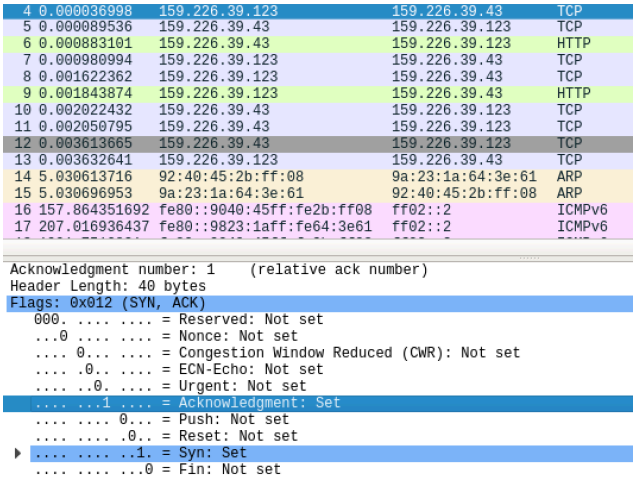

图 4 h2 收到 h1 的连接接收信号

图 2 中，h2 首先广播 ARP 请求，获取到了 h1 的地址；然后（图 5）发送 TCP

请求（SYN=1，ACK=0）请求与 h1 建立连接；在收到 h1 的 ACK 信号（图 4）后，向 h1 发送 ACK 确认，之后通过 http 协议来获取 html 的信息；接着通过 TCP 协议进行数据的传输。

图 3 中，h1 收到来自 h2 的 ARP 请求后，向 h2 单播了自己的地址；来自 h2 的 tcp 请求经过 NAT 转换后，被 h1 接收；h1 向 h2 发送 ACK 消息，经 NAT 转换后，到达 h2；之后，h1 再次接收到了 h2 经 NAT 转换后的 ACK 确认，这时 tcp 连接成功建立。然后开始传输数据。

（二）代码实现分析

⇨ `get_packet_direction(char *packet)`

该函数通过数据包中的**目的地址和源地址**来判断包的发送方向。

```
static int get_packet_direction(char *packet)
{
    struct iphdr *ip = packet_to_ip_hdr(packet);
    u32 saddr = ntohl(ip->saddr);
    u32 daddr = ntohl(ip->daddr);

    if(IS_DIR_OUT(saddr,daddr))
        return DIR_OUT;
    else if(IS_DIR_IN(saddr,daddr))
        return DIR_IN;
    else
        return DIR_INVALID;
}
```

判断逻辑如下：

```
#define IS_INTERNAL_IP(addr) \
    (longest_prefix_match((addr))->iface->ip == nat.internal_iface->ip)
#define IS_EXTERNAL_IP(addr) \
    (longest_prefix_match((addr))->iface->ip == nat.external_iface->ip)

#define IS_DIR_IN(saddr,daddr) \
    (daddr == nat.external_iface->ip) && IS_EXTERNAL_IP(saddr)
#define IS_DIR_OUT(saddr,daddr) \
    IS_INTERNAL_IP(saddr) && IS_EXTERNAL_IP(daddr)
```

如果路由表中地址和 NAT 的内网端口地址一致，则判定为内部 IP，反之为外部 IP；如果目的地址是外部 IP，源地址是内部 IP，则判定方向为 IN；如果目的地址为 NAT 的外网端口地址，源地址是外部 IP，则判定方向为 OUT。

⇨ do_translation(iface_info_t *iface, char *packet, int len, int dir)

```
void do_translation(iface_info_t *iface, char *packet, int len, int dir){
    struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
    struct tcphdr *tcp_hdr = packet_to_tcp_hdr(packet);
    u16 sport = ntohs(tcp_hdr->sport);
    u32 saddr = ntohl(ip_hdr->saddr);
    u16 dport = ntohs(tcp_hdr->dport);
    u32 daddr = ntohl(ip_hdr->daddr);
    // find out whether there exists nat mapping
    u32 serv_ip = (dir == DIR_IN) ? saddr : daddr;
    u16 serv_port = (dir == DIR_IN) ? sport : dport;
    u8  hash_value = caculate_hash8(serv_port, serv_ip);
    struct list_head * mapping_entry = &(nat.nat_mapping_list[hash_value]);
    pthread_mutex_lock(&nat.lock);
    struct nat_mapping * pos = NULL, *q = NULL;
    if(!list_empty(mapping_entry)){
        list_for_each_entry_safe(pos, q, mapping_entry, list){
            if(dir == DIR_OUT && NAT_MAPPING_MATCH_IN(pos,saddr,sport)){
                printf("OUT: ip and port mapping matches.\n");
                update_ip_and_tcp_header(packet,
                                         nat.external_iface->ip,
                                         pos->external_port,
                                         DIR_OUT);
                pos->update_time = time(NULL);
                ip_send_packet(packet, len);
                recover_unused_conn(pos, tcp_hdr);
                pthread_mutex_unlock(&nat.lock);
                return ;
            }
            else if(dir == DIR_IN && NAT_MAPPING_MATCH_EX(pos,daddr,dport)){
                printf("IN: ip and port mapping matches.\n");
                update_ip_and_tcp_header(packet,
                                         pos->internal_ip,
                                         pos->internal_port,
                                         DIR_IN);
                pos->update_time = time(NULL);
                ip_send_packet(packet, len);
                recover_unused_conn(pos, tcp_hdr);
                pthread_mutex_unlock(&nat.lock);
                return ;
            }
        }
    }
    // nat mapping does not find (OUT)
    // assign a new port and build a new connection
    u16 new_port = assign_external_port();
    struct nat_mapping *new_mapping = init_new_mapping(saddr,sport,new_port);
    list_add_tail(&(new_mapping->list), mapping_entry);
    // update saddr, sport and checksum of tcp header and ip header
    update_ip_and_tcp_header(packet,
                             (nat.external_iface)->ip,
                             new_port,
                             DIR_OUT);
    pthread_mutex_unlock(&nat.lock);
    ip_send_packet(packet, len);
    return ;
}
```

这是本次实验的核心代码,用于 NAT 的网络地址转换,包含了 NAT 表的添加、查找、更新操作。

先根据（serv_ip， serv_port）二元组的值计算查找哈希表的哈希值;

```
u8 caculate_hash8(u16 serv_port, u32 serv_ip){
    char buf[6];
    memcpy(buf, &serv_ip, 4);
    memcpy(buf+4, &serv_port, 2);
    return hash8(buf, 6);
}
```

然后得到对应 nat mapping 的链表;
如果链表为空，显然需要插入新的 nat mapping;

```
    u16 new_port = assign_external_port();
    struct nat_mapping *new_mapping = init_new_mapping(saddr,sport,new_port);
    list_add_tail(&(new_mapping->list), mapping_entry);
    // update saddr, sport and checksum of tcp header and ip header
    update_ip_and_tcp_header(packet,
                             (nat.external_iface)->ip,
                             new_port,
                             DIR_OUT);
```

否则，根据包的方向和(ip,port)二元组来查找对应的 nat mapping。
查找到对应的 nat mapping 后，更新 tcp 和 ip 头部，并刷新该 mapping 的更新时间，然后将包通过 ip_send_packet()发送出去。

```
void update_ip_and_tcp_header(char *packet, u32 addr, u16 port, int dir){
    struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
    struct tcphdr *tcp_hdr = packet_to_tcp_hdr(packet);
    if(dir == DIR_IN){
        ip_hdr->daddr  = htonl(addr);
        tcp_hdr->dport = htons(port);
    }
    else{
        ip_hdr->saddr  = htonl(addr);
        tcp_hdr->sport = htons(port);
    }
    tcp_hdr->checksum = tcp_checksum(ip_hdr, tcp_hdr);
    ip_hdr->checksum  = ip_checksum(ip_hdr);
}
```

跟新头部，即根据方向修改相应的部分：

1. 方向为 IN, 更新 ip->daddr 和 tcp->dport;
2. 方向为 OUT,更新 ip->saddr 和 tcp->sport;