网络实验报告

雷正宇 2016K8009909005

实验内容

SNAT实验

运行给定网络拓扑(nat_topo.py)

在n1, h1, h2, h3上运行相应脚本

在n1上运行nat程序： n1# ./nat

在h3上运行HTTP服务：h3# python ./http_server.py

在h1, h2上分别访问h3的HTTP服务

h1# wget http://159.226.39.123:8000

h2# wget http://159.226.39.123:8000

DNAT实验

运行给定网络拓扑(nat_topo.py)

在n1, h1, h2, h3上运行相应脚本

在n1上运行nat程序： n1# ./nat

在h1, h2上分别运行HTTP Server： h1/h2# python ./http_server.py

在h3上分别请求h1, h2页面

h3# wget http://159.226.39.43:8000

h3# wget http://159.226.39.43:8001

手动构造一个包含两个nat的拓扑

h1 <-> n1 <-> n2 <-> h2

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

实验流程

判断数据包方向的过程

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

```c
static int get_packet_direction(char *packet)
{
  struct iphdr *ip = packet_to_ip_hdr(packet);
  iface_info_t *siface = longest_prefix_match(ntohl(ip->saddr))->iface;
  iface_info_t *diface = longest_prefix_match(ntohl(ip->daddr))->iface;
  if(siface == nat.internal_iface && diface == nat.external_iface)
    return DIR_OUT;

  if(siface == nat.external_iface && ntohl(ip->daddr) ==
nat.external_iface->ip)
    return DIR_IN;

  return DIR_INVALID;
}
```

更新数据包头以及发送过程

```c
void trans_and_send_packet(char *packet, struct nat_mapping *mapping, int
len, int dir)
{
  struct iphdr  *ip_hdr  = packet_to_ip_hdr(packet);
  struct tcphdr *tcp_hdr = packet_to_tcp_hdr(packet);

  if(tcp_hdr->flags & TCP_FIN)
    mapping->conn.external_fin = 1;
  if(tcp_hdr->flags & TCP_ACK)
    mapping->conn.external_ack = 1;
  if(tcp_hdr->flags & TCP_RST) {
    mapping->conn.external_fin = 1;
    mapping->conn.external_ack = 1;
    mapping->conn.internal_fin = 1;
    mapping->conn.internal_ack = 1;
  }

  if(dir == DIR_OUT) {
    ip_hdr->saddr = htonl(nat.external_iface->ip);
    tcp_hdr->sport = htons(mapping->external_port);
  }
  else if (dir == DIR_IN) {
    ip_hdr->daddr = htonl(mapping->internal_ip);
    tcp_hdr->dport = htons(mapping->internal_port);
  }
  tcp_hdr->checksum = tcp_checksum(ip_hdr, tcp_hdr);
  ip_hdr->checksum = ip_checksum(ip_hdr);
  ip_send_packet(packet, len);
}
```

这个函数应当是处理翻译过程的一部分，剥离出来完全是因为原函数太长。

完整转换过程

转换的逻辑如下：

```
if dir == DIR_OUT
  查找对应mapping
    如果不存在，建立新的mapping
  转发包
else if dir == DIR_IN
  查找对应mapping
    如果不存在，根据rules建立
  更新数据包头并发送
else
  回复ICMP Destination Host Unreachable
```

具体实现太长，就不粘贴代码了。其中用到了在讲义中抽象出来的 assign_external_port(), 实现如下：

```
u16 assign_external_port()
{
  for (u16 i = NAT_PORT_MIN; i < NAT_PORT_MAX; i ++) {
    if (!nat.assigned_ports[i]) {
      nat.assigned_ports[i] = 1;
      return i;
    }
  }
  return -1;
}
```

老化操作

双方都已发送FIN且回复相应ACK的连接，一方发送RST包的连接，可以直接回收

双方已经超过60秒未传输数据的连接，认为其已经传输结束，可以回收

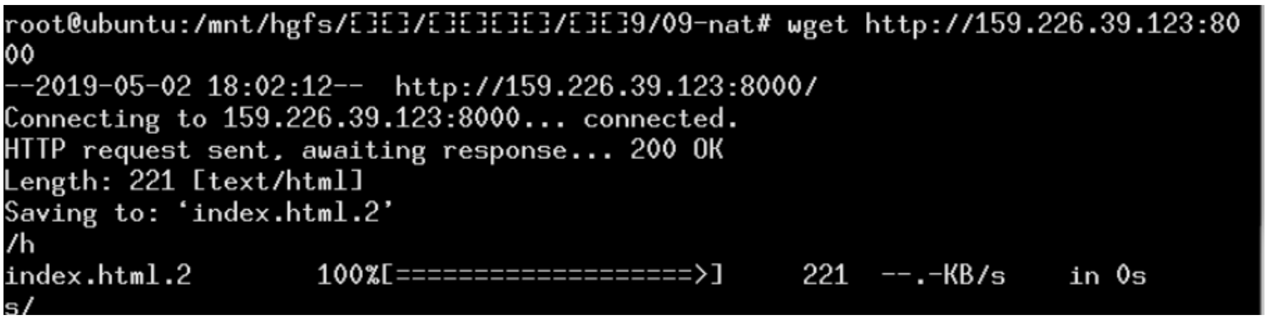
```
void *nat_timeout()
{
  while (1) {
    pthread_mutex_lock(&nat.lock);
    for (int i = 0; i < HASH_8BITS; i ++) {
      struct list_head *head = &nat.nat_mapping_list[i];
      struct nat_mapping *mapping_p, *entry;
      list_for_each_entry_safe(mapping_p, entry, head, list) {
        mapping_p->update_time += 1;
        if (time(NULL) - mapping_p->update_time > TCP_ESTABLISHED_TIMEOUT
|| is_flow_finished(&mapping_p->conn)) {
          nat.assigned_ports[mapping_p->external_port] = 0;
          list_delete_entry(&mapping_p->list);
          free(mapping_p);
        }
      }
    }
    pthread_mutex_unlock(&nat.lock);
    sleep(1);
  }
  return NULL;
}
```

NAT退出

清除mapping并执行老化即可，过程简单。

实验结果及分析

SNAT实验

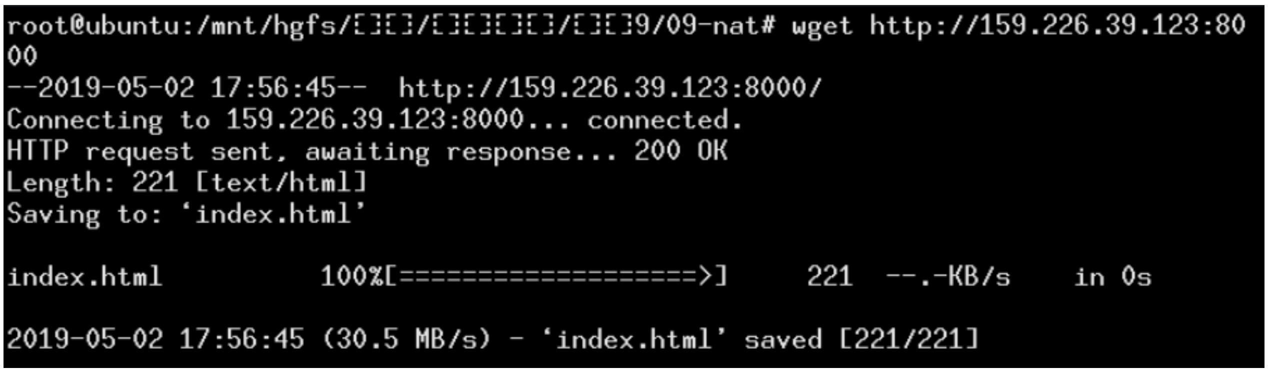


```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# wget http://159.226.39.123:80
00
--2019-05-02 18:02:12--  http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 221 [text/html]
Saving to: ‘index.html.2’
/h
index.html.2        100%[===================>]     221  --.-KB/s    in 0s
s/
```

h1获得html文件如下：

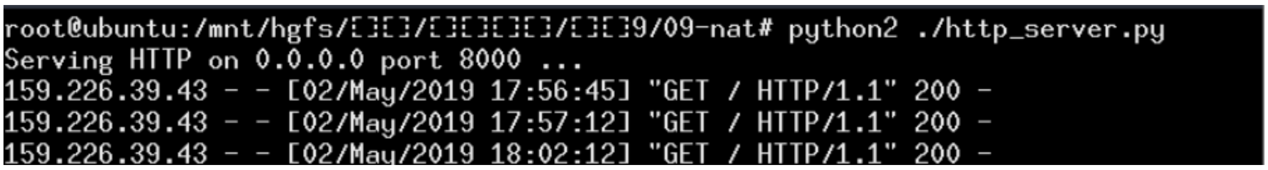
```
<!doctype html>
<html>
  <head> <meta charset="utf-8">
    <title>Network IP Address</title>
  </head>
  <body>
            My network IP is: 159.226.39.123

            Remote IP is: 159.226.39.43
        </body>
</html>
```

h2获得的html和h1一样，过程基本相同：



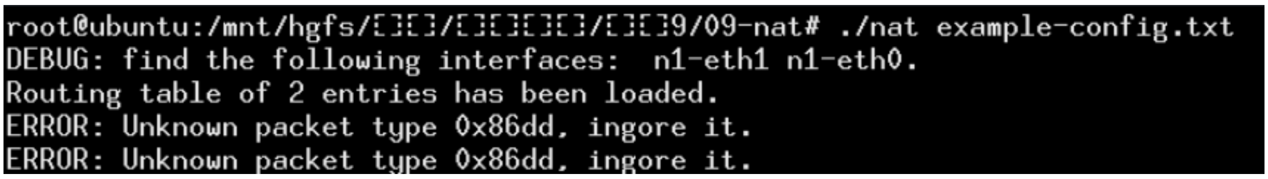
```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# wget http://159.226.39.123:80
00
--2019-05-02 17:56:45--  http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 221 [text/html]
Saving to: ‘index.html’

index.html          100%[===================>]     221  --.-KB/s    in 0s

2019-05-02 17:56:45 (30.5 MB/s) - ‘index.html’ saved [221/221]
```

h3提示信息如下：



```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# python2 ./http_server.py
Serving HTTP on 0.0.0.0 port 8000 ...
159.226.39.43 - - [02/May/2019 17:56:45] "GET / HTTP/1.1" 200 -
159.226.39.43 - - [02/May/2019 17:57:12] "GET / HTTP/1.1" 200 -
159.226.39.43 - - [02/May/2019 18:02:12] "GET / HTTP/1.1" 200 -
```

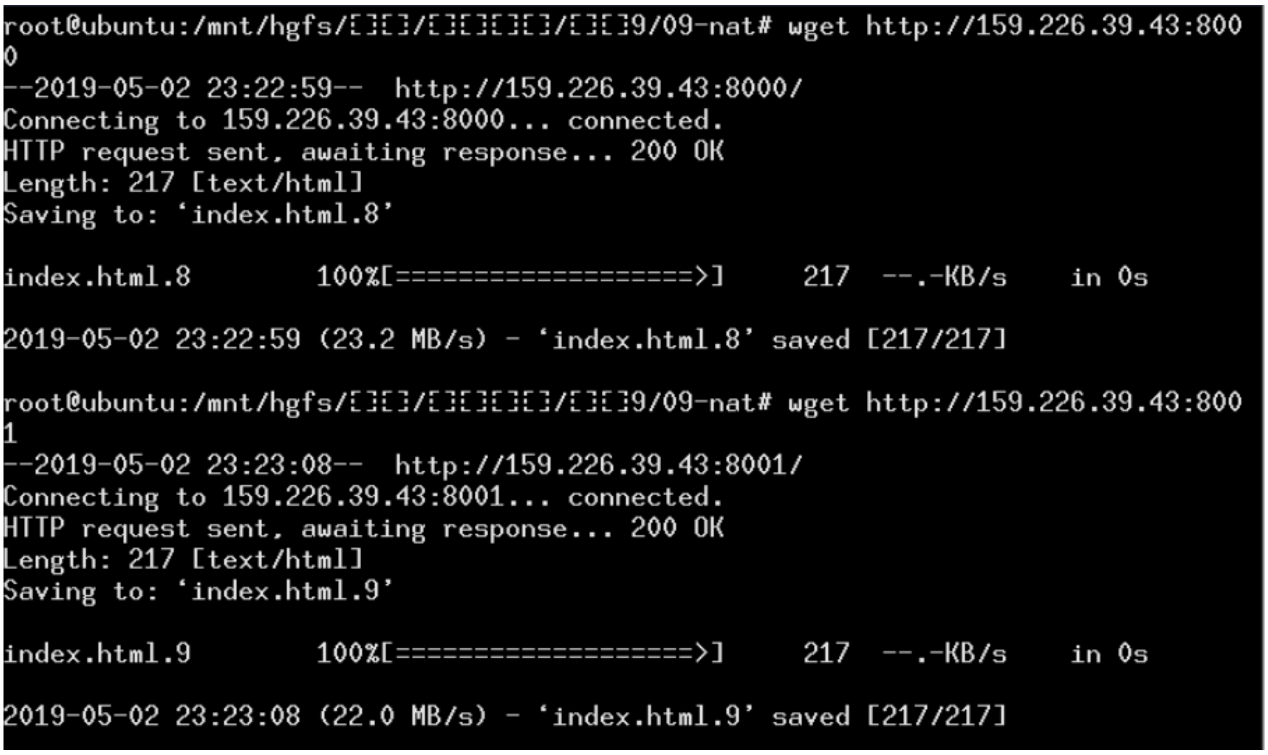
n1会不时报出这样的错误，通过wireshark查看，推测可能是操作系统发出的无用包（尚未确定）。



```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# ./nat example-config.txt
DEBUG: find the following interfaces:  n1-eth1 n1-eth0.
Routing table of 2 entries has been loaded.
ERROR: Unknown packet type 0x86dd, ingore it.
ERROR: Unknown packet type 0x86dd, ingore it.
```

DNAT实验

h3访问h1和h2的http服务过程如下：



```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# wget http://159.226.39.43:800
0
--2019-05-02 23:22:59--  http://159.226.39.43:8000/
Connecting to 159.226.39.43:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 217 [text/html]
Saving to: ‘index.html.8’

index.html.8        100%[===================>]     217  --.-KB/s    in 0s

2019-05-02 23:22:59 (23.2 MB/s) - ‘index.html.8’ saved [217/217]

root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# wget http://159.226.39.43:800
1
--2019-05-02 23:23:08--  http://159.226.39.43:8001/
Connecting to 159.226.39.43:8001... connected.
HTTP request sent, awaiting response... 200 OK
Length: 217 [text/html]
Saving to: ‘index.html.9’

index.html.9        100%[===================>]     217  --.-KB/s    in 0s

2019-05-02 23:23:08 (22.0 MB/s) - ‘index.html.9’ saved [217/217]
```

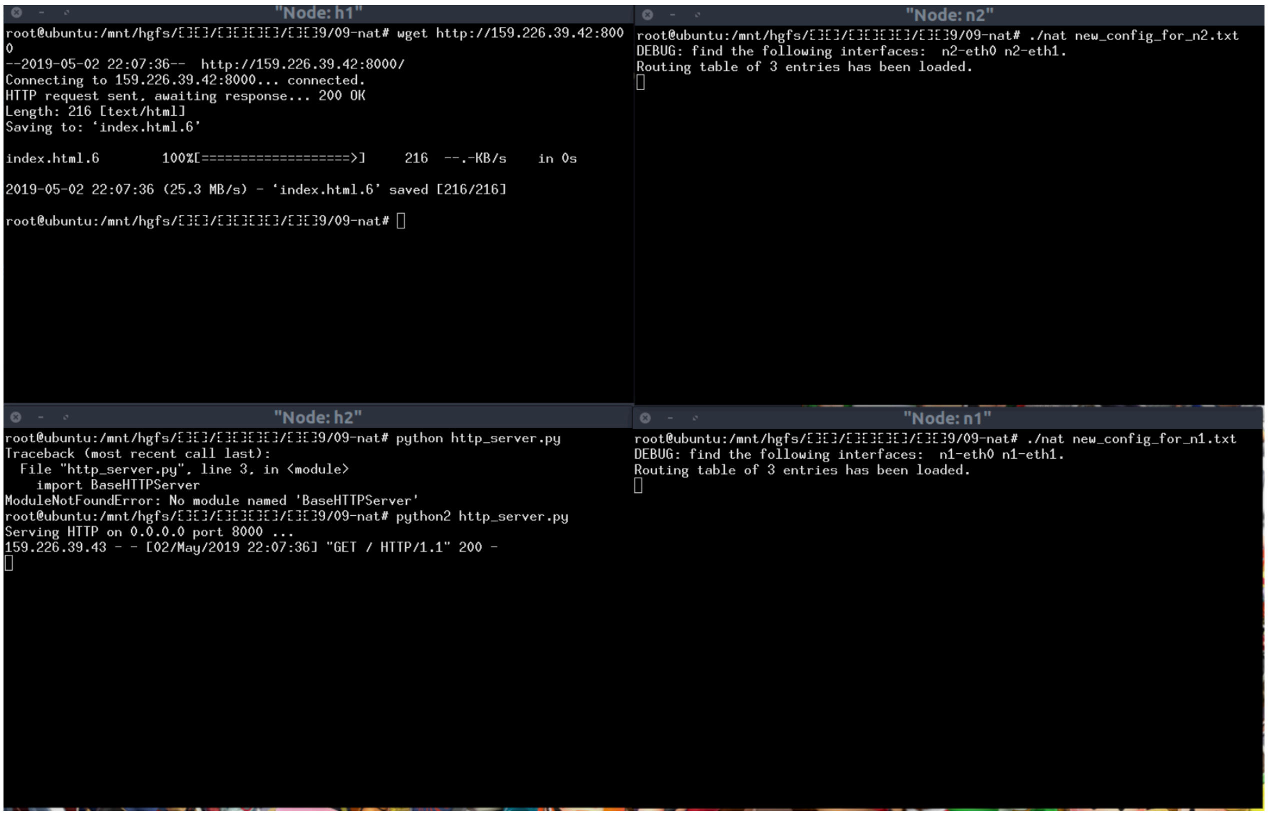
得到的html回应和SNAT实验相同。

手动构造拓扑

构造h1 - n1 - n2 - h2拓扑的代码如下：

```python
#!/usr/bin/python

from mininet.node import OVSBridge
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class NATTopo(Topo):
    def build(self):
        h1 = self.addHost('h1')
        n1 = self.addHost('n1')
        n2 = self.addHost('n2')
        h2 = self.addHost('h2')

        self.addLink(h1, n1)
        self.addLink(n1, n2)
        self.addLink(n2, h2)

if __name__ == '__main__':
    topo = NATTopo()
    net = Mininet(topo = topo, switch = OVSBridge, controller = None)

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

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

    n1.cmd('ifconfig n1-eth0 10.21.0.254/16')
    n1.cmd('ifconfig n1-eth1 159.226.39.43/24')
    n1.cmd('route add default gw 159.226.39.42')

    n2.cmd('ifconfig n2-eth0 159.226.39.42/24')
    n2.cmd('ifconfig n2-eth1 10.21.0.253/16')
    n2.cmd('route add default gw 159.226.39.43')

    h2.cmd('ifconfig h2-eth0 10.21.0.2/16')
    h2.cmd('route add default gw 10.21.0.253')

    for h in (h1, h2):
        h.cmd('./scripts/disable_offloading.sh')
        h.cmd('./scripts/disable_ipv6.sh')

    for n in (n1, n2):
        n.cmd('./scripts/disable_arp.sh')
        n.cmd('./scripts/disable_icmp.sh')
        n.cmd('./scripts/disable_ip_forward.sh')
        n.cmd('./scripts/disable_ipv6.sh')

    net.start()
    CLI(net)
    net.stop()
```

运作结果如下：



"Node: h1"

```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# wget http://159.226.39.42:800
0
--2019-05-02 22:07:36--  http://159.226.39.42:8000/
Connecting to 159.226.39.42:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 216 [text/html]
Saving to: ‘index.html.6’

index.html.6        100%[===================>]     216  --.-KB/s    in 0s

2019-05-02 22:07:36 (25.3 MB/s) - ‘index.html.6’ saved [216/216]

root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# 
```

"Node: n2"

```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# ./nat new_config_for_n2.txt
DEBUG: find the following interfaces:  n2-eth0 n2-eth1.
Routing table of 3 entries has been loaded.
```

"Node: h2"

```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# python http_server.py
Traceback (most recent call last):
  File "http_server.py", line 3, in <module>
    import BaseHTTPServer
ModuleNotFoundError: No module named 'BaseHTTPServer'
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# python2 http_server.py
Serving HTTP on 0.0.0.0 port 8000 ...
159.226.39.43 - - [02/May/2019 22:07:36] "GET / HTTP/1.1" 200 -
```

"Node: n1"

```
root@ubuntu:/mnt/hgfs/[][]/[][][][]/[][]9/09-nat# ./nat new_config_for_n1.txt
DEBUG: find the following interfaces:  n1-eth0 n1-eth1.
Routing table of 3 entries has been loaded.
```

这里有一个小坑，就是源文件给出的parse函数，需要做一点小的修改，将默认的n1-eth0和n1-eth1换成词法分析出的string.