东南大学 SOUTHEAST UNIVERSITY | 网络空间安全学院 School of Cyber Science and Engineering

网络空间安全综合课程设计

实验报告（七）

学号：57117137 姓名：刘康亮

东南大学网络空间安全学院

2020 年 9 月 26 日

VPN Tunneling Lab

Task1

虚拟机 A(seed)一张网卡，桥接至宿主机无线网卡

虚拟机 B（普通 ubuntu）两张网卡，一张桥接至宿主机无线网卡，另一张连接至内部网络（内部网络名称为 intnet）并配置 IP 为 10.0.0.1/24，网关也为 10.0.0.1

虚拟机 C（security onion）一张网卡连接至内部网络（内部网络名称为 intnet）并配置 IP 为 10.0.0.2/24，网关为 10.0.0.1

虚拟机 A 和 B 桥接至宿主机的网卡 ip 为自动分配，不过都在 192.168.1.0/24 内。

B ping C:

```
nie@nie-VirtualBox:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.382 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.310 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.284 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 0.284/0.325/0.382/0.044 ms
nie@nie-VirtualBox:~$
```

A ping C

```
[09/21/20]seed@VM:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6142ms

[09/21/20]seed@VM:~$
```

Task2

Task2.a

修改示例程序为:

```
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
```

运行后:

```
[09/21/20]seed@VM:~/Lab/lab7$ sudo ./tun.py
Interface Name: nie0
```

```
5: nie0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc
ult qlen 500
    link/none
[09/21/20]seed@VM:~$
```

Task2.b

ip 设置为 192.168.153.88/24

```
5: nie0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qd
te UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.88/24 scope global nie0
       valid_lft forever preferred_lft forever
    inet6 fe80::32f:73d7:c461:105c/64 scope link flags 800
       valid_lft forever preferred_lft forever
[09/21/20]seed@VM:~$
```

Task2.c

尝试 ping192.168.53.87:

```
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 84
  id        = 22787
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = 0xf5a5
  src       = 192.168.53.88
  dst       = 192.168.53.87
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0x96eb
     id        = 0x1424
     seq       = 0x5
###[ Raw ]###
        load      = '\x06li \xf0\x1c\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x
```

尝试 ping 10.0.0.2，什么都没有显示

Task2.d

程序如下时：

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                ip = IP(packet)
                ip.show()
                newip = IP(src='1.2.3.4', dst='192.168.53.88')
                newpkt = newip/ip.payload
                os.write(tun, bytes(newpkt))
```

即收到一个包，以 1.2.3.4 为源向 tun（192.168.53.88）发送一个包
终端 ping 192.168.53.87，wireshark 如下：

3	2020-09-23 03:41:04.958469217	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) request id=0x0e43, seq=1/256,
4	2020-09-23 03:41:04.960154937	1.2.3.4	192.168.53.88	ICMP	84	Echo (ping) request id=0x0e43, seq=1/256,
5	2020-09-23 03:41:05.969690217	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) request id=0x0e43, seq=2/512,
6	2020-09-23 03:41:05.971238429	1.2.3.4	192.168.53.88	ICMP	84	Echo (ping) request id=0x0e43, seq=2/512,
7	2020-09-23 03:41:06.997389177	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) request id=0x0e43, seq=3/768,
8	2020-09-23 03:41:06.998817362	1.2.3.4	192.168.53.88	ICMP	84	Echo (ping) request id=0x0e43, seq=3/768,
9	2020-09-23 03:41:08.038931172	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) request id=0x0e43, seq=4/1024
10	2020-09-23 03:41:08.040342833	1.2.3.4	192.168.53.88	ICMP	84	Echo (ping) request id=0x0e43, seq=4/1024

程序改为如下：

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                ip = IP(packet)
                ip.show()
                newpkt = "eeeeeeeeeeee".encode('utf-8')
                os.write(tun, bytes(newpkt))
```

即写入 12 个 e
运行程序并 ping192.168.53.87
Wireshark：

3	2020-09-23 03:45:31.835111963	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) reque
4	2020-09-23 03:45:31.836029942	N/A	N/A	IPv6	12	Invalid IPv6 head
5	2020-09-23 03:45:32.847756750	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) reque
6	2020-09-23 03:45:32.848624354	N/A	N/A	IPv6	12	Invalid IPv6 head
7	2020-09-23 03:45:33.902507895	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) reque
8	2020-09-23 03:45:33.903259452	N/A	N/A	IPv6	12	Invalid IPv6 head
9	2020-09-23 03:45:34.927340858	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) reque
10	2020-09-23 03:45:34.928015520	N/A	N/A	IPv6	12	Invalid IPv6 head
11	2020-09-23 03:45:35.951189229	192.168.53.88	192.168.53.87	ICMP	84	Echo (ping) reque
12	2020-09-23 03:45:35.951853436	N/A	N/A	IPv6	12	Invalid IPv6 head

```
00  65 65 65 65 65 65 65 65  65 65 65 65                eeeeeeee eeee
```

可以收到包，还是 12 个 e，没有添加任何头。

Task3
Server 端程序使用实验指导的即可
Client 程序如下：

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                sock.sendto(packet, ('192.168.1.102', 9090))
```

192.168.1.102 为 vpn server 所在虚拟机的对外的网卡地址

两边执行程序后，在 client 端虚拟机上 ping192.168.53.87，server 端如下：

```
192.168.1.104:38577 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 192.168.53.87
192.168.1.104:38577 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 192.168.53.87
192.168.1.104:38577 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 192.168.53.87
```

报告一开始说明了实验中内网地址分配为 10.0.0.0/24

所以在 client 中添加路由表项：

```
[09/23/20]seed@VM:~$ sudo ip route add 10.0.0.0/24  via 192.168.53.88
```

192.168.53.88 即为 tun 卡的 ip

再 ping10.0.0.2

server 端如下：

```
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
```

成功收到

Task4：

因为以之前的三台虚拟机的配置（与实验指导的有区别）总是无法成功，在没有头绪的情况下选择了完全按照实验指导手册重新设置三台虚拟机。

新的配置如下

三台 seed

seed 与 seed3（Host V）连接至同一内部网络，ip 分别为 192.168.60.1/24，192.168.60.2/24

seed 与 seed2（Host U）连接至同一 NAT 网络（模拟外网），ip 分别为 10.0.2.4/24，10.0.2.7/24

然后就是最关键的，关于 seed2 和 seed 上面的两张 tun 网卡，经过一些小实验之后我得到了如下条件，即这两张网卡需要在同一网段内且不能同 ip 实验才能成功，即 seed 上的 tun 卡会把包传给内核，内核再传至连接内部网络的网卡再传至另一主机。主要是对 tun 卡的工作原理不太清楚，如果同 ip 的话似乎包根本没有经过我们的程序（程序中的输出部分根本没有输出），也就是在更下层就处理掉了，如果不是同网段倒是经过我们的 tun 程序，但不会传给内核（似乎是）。以上是一些猜想。

seed 上的 tun 卡 ip 为 192.168.53.87/24，seed2 上的 tun 卡 ip 为.88/24

tun_server.py 如下：

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.87/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
IP_A = "10.0.2.4"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
        data, (ip, port) = sock.recvfrom(2048)
        print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
        pkt=IP(data)
        print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
        os.write(tun, data)
```

tun.py（seed2 上的客户端程序）如下：

```python
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))
print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                print(1)
                sock.sendto(packet, ('10.0.2.4', 9090))
```

把配置路由表的语句也写入程序，这样就不用再输命令配置了。

在 seed2 （Host U）ping seed3（Host V）后

Seed 上：

```
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
```

Host V 上：

→	1	2020-09-25 23:03:56.060402316	192.168.53.88	192.168.60.2	ICMP	98 Echo (ping) request
←	2	2020-09-25 23:03:56.060471302	192.168.60.2	192.168.53.88	ICMP	98 Echo (ping) reply
	3	2020-09-25 23:03:57.084413256	192.168.53.88	192.168.60.2	ICMP	98 Echo (ping) request
	4	2020-09-25 23:03:57.084432374	192.168.60.2	192.168.53.88	ICMP	98 Echo (ping) reply
	5	2020-09-25 23:03:58.109730914	192.168.53.88	192.168.60.2	ICMP	98 Echo (ping) request
	6	2020-09-25 23:03:58.109755354	192.168.60.2	192.168.53.88	ICMP	98 Echo (ping) reply
	7	2020-09-25 23:03:59.133509046	192.168.53.88	192.168.60.2	ICMP	98 Echo (ping) request
	8	2020-09-25 23:03:59.133534056	192.168.60.2	192.168.53.88	ICMP	98 Echo (ping) reply
	9	2020-09-25 23:04:00.156899624	192.168.53.88	192.168.60.2	ICMP	98 Echo (ping) request
	10	2020-09-25 23:04:00.156954650	192.168.60.2	192.168.53.88	ICMP	98 Echo (ping) reply
	13	2020-09-25 23:04:01.183123091	192.168.53.88	192.168.60.2	ICMP	98 Echo (ping) request
	14	2020-09-25 23:04:01.183147164	192.168.60.2	192.168.53.88	ICMP	98 Echo (ping) reply

看到了 U（53.88）传至 V（60.2）的包