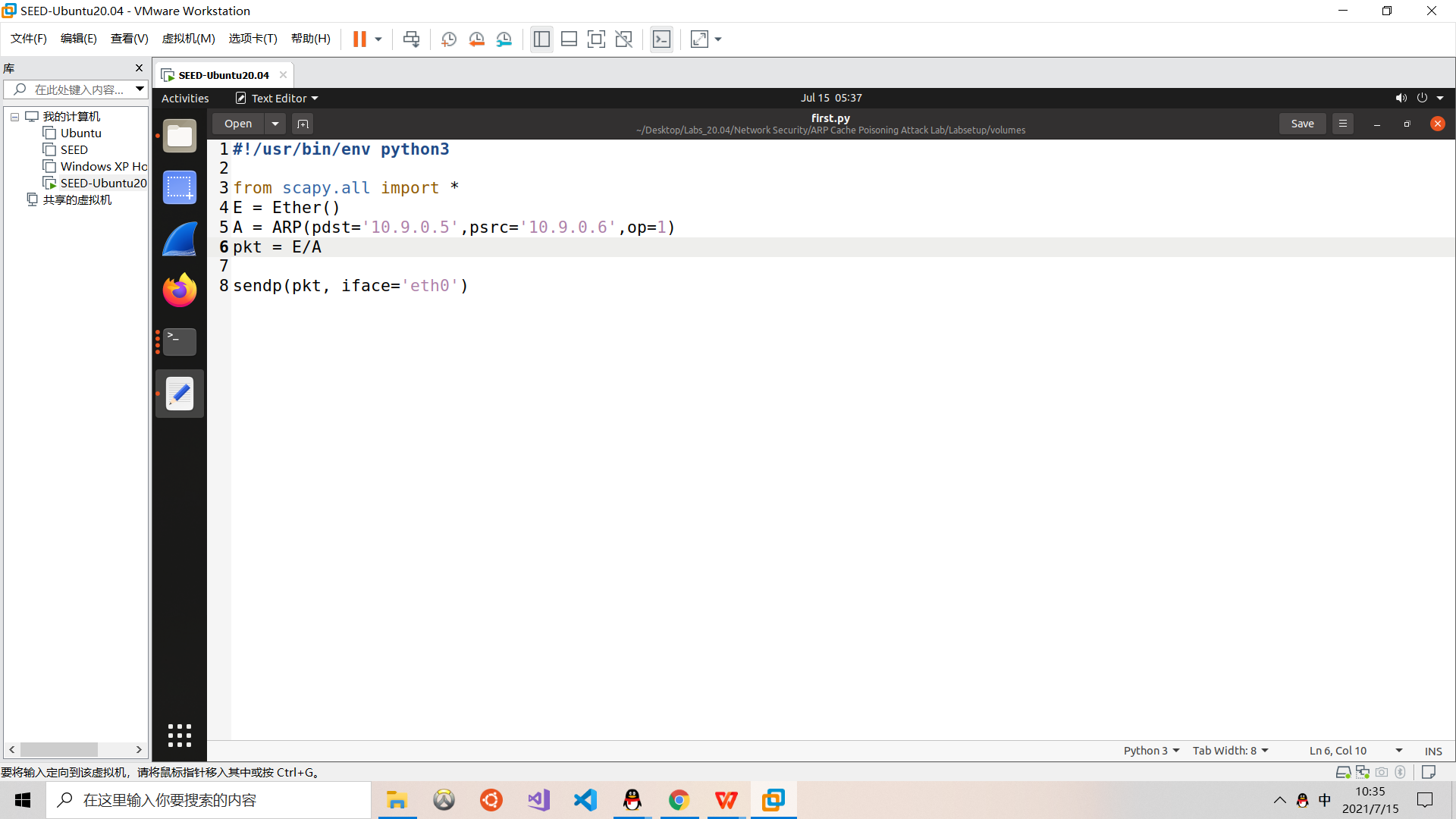
**第四次实验**

57118122李泽厚

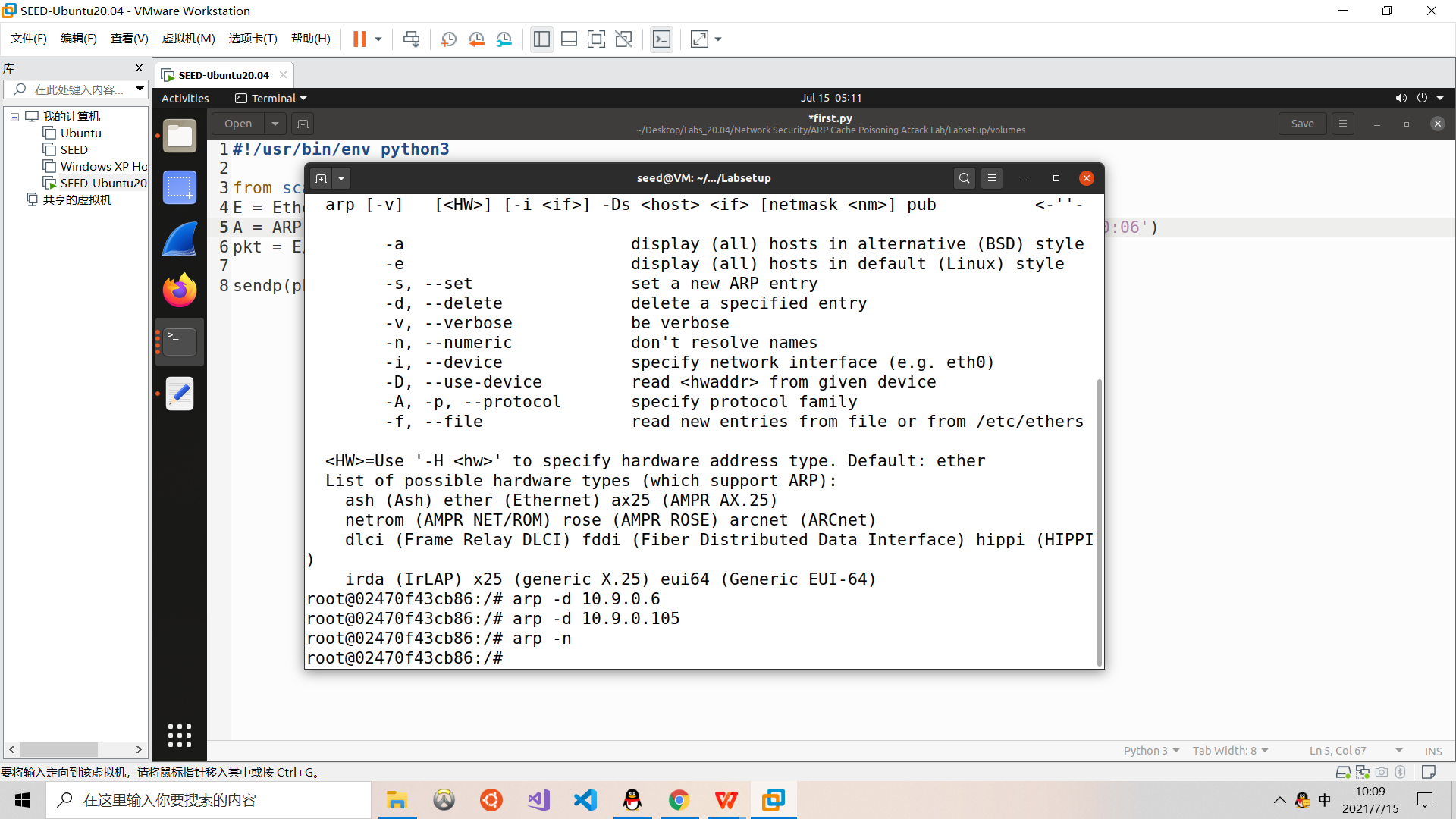
**Task 1: ARP Cache Poisoning**

1. task1.A：

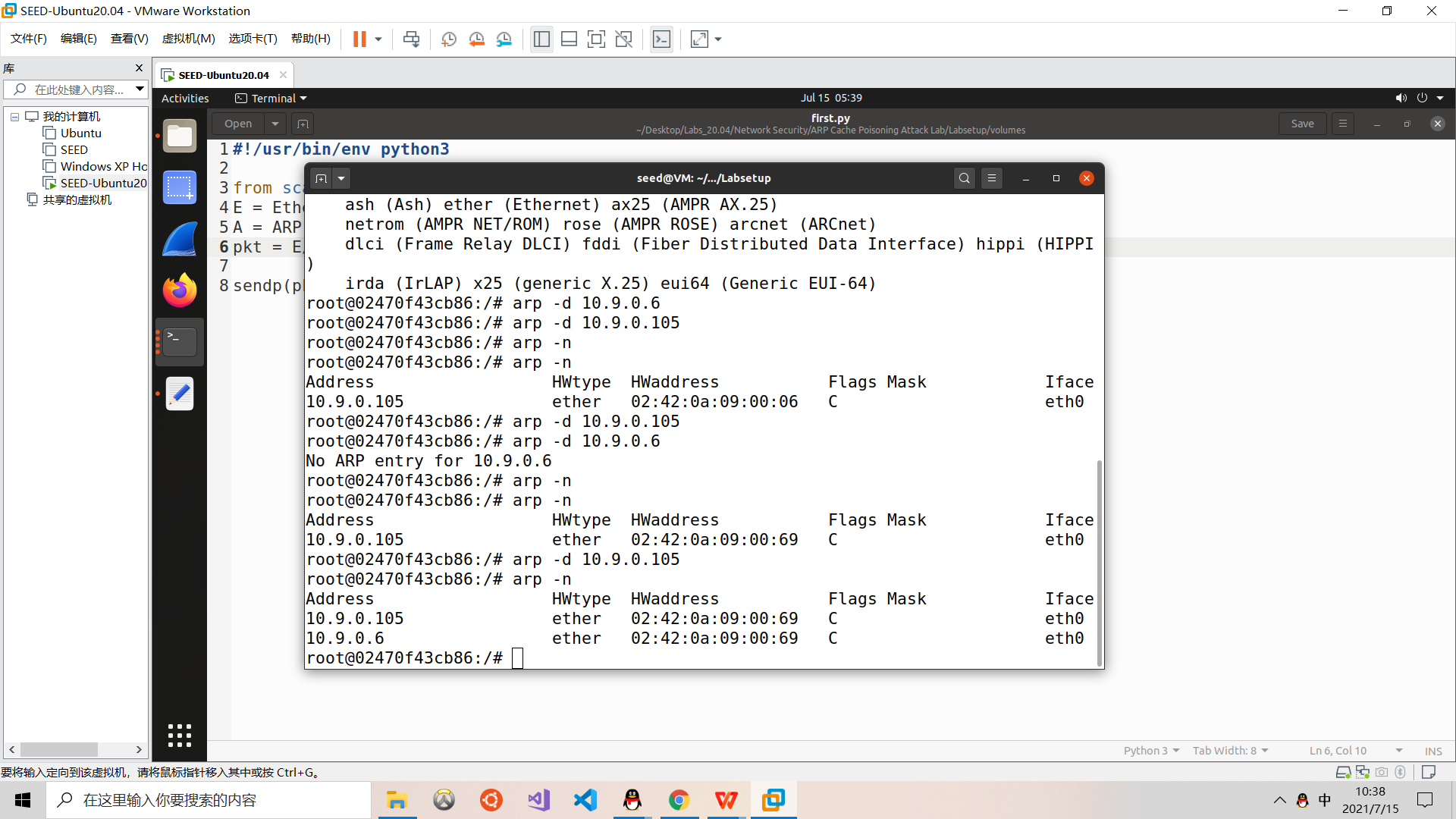
在共享文件夹下写下如下脚本：



进入10.9.0.5，输入命令 arp -n后并没有结果：



然后我们进入10.9.0.105，运行上面的脚本。这时候再在10.9.0.5上输入arp - n，显示如下结果：

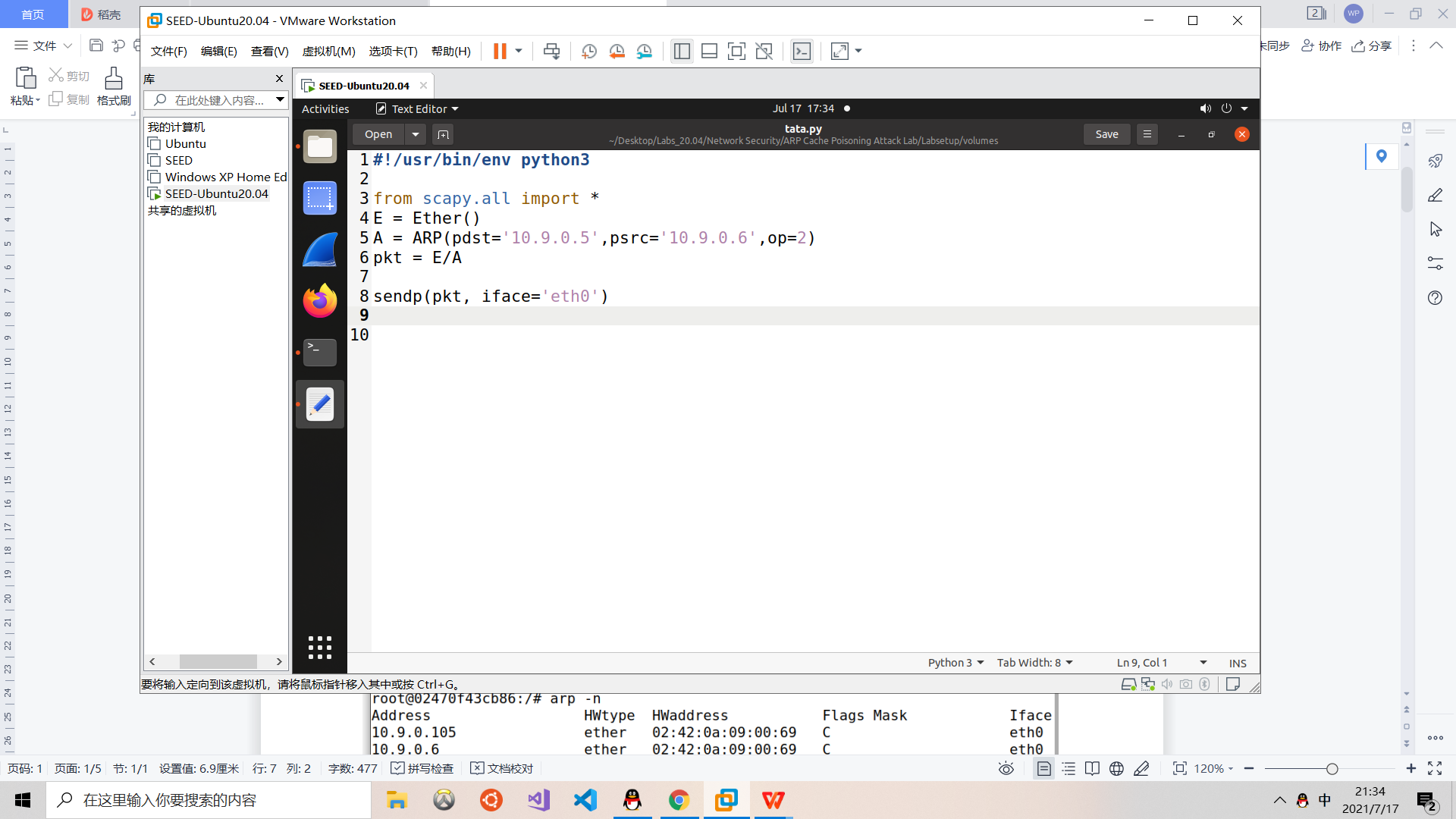


如上图所示，在A的arp缓存里，B的IP地址映射到了M的MAC地址。

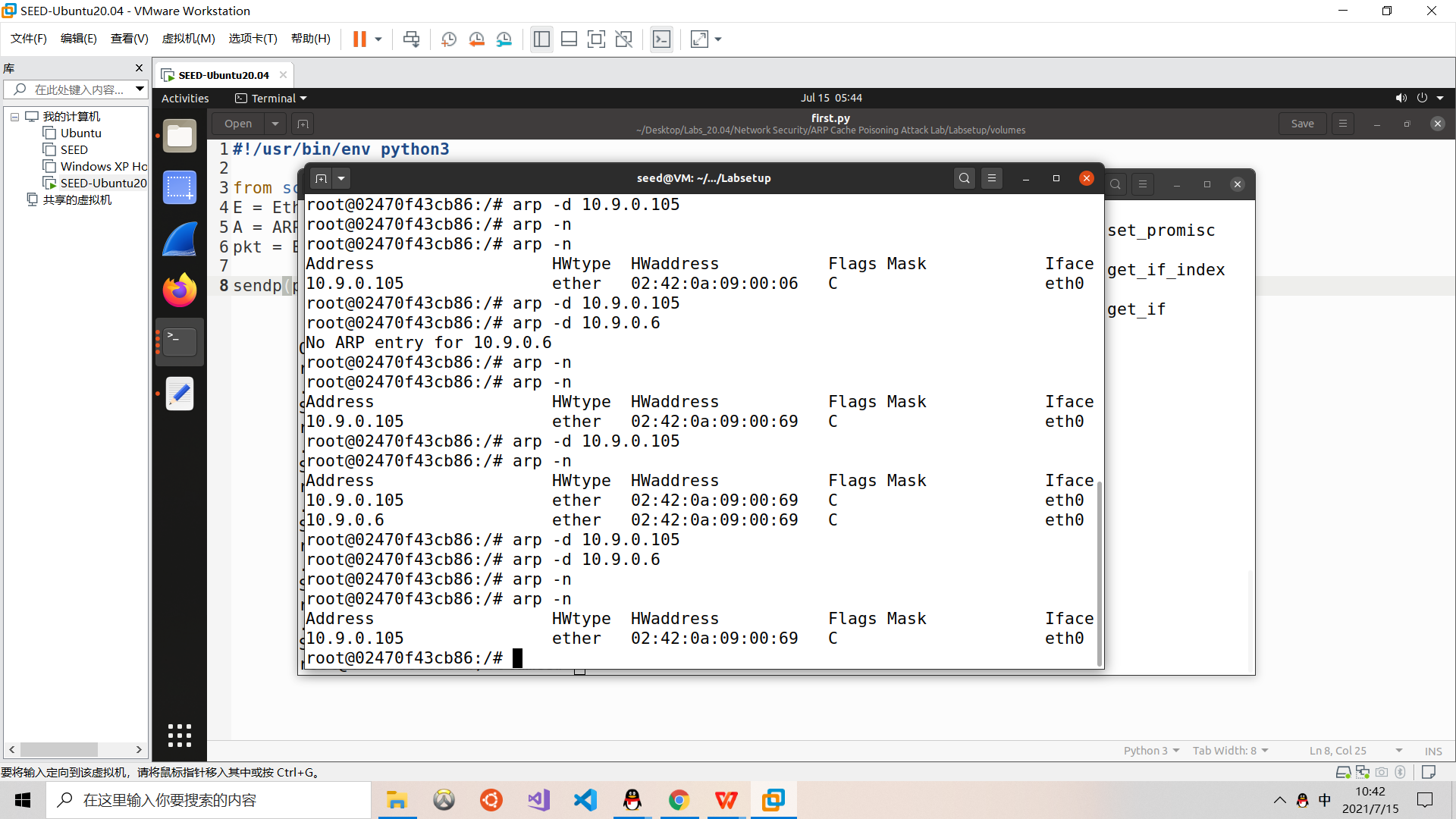
由于arp协议本质上是用IP地址找mac地址，之后A发给B的信息都会发给M。

1. task1.B

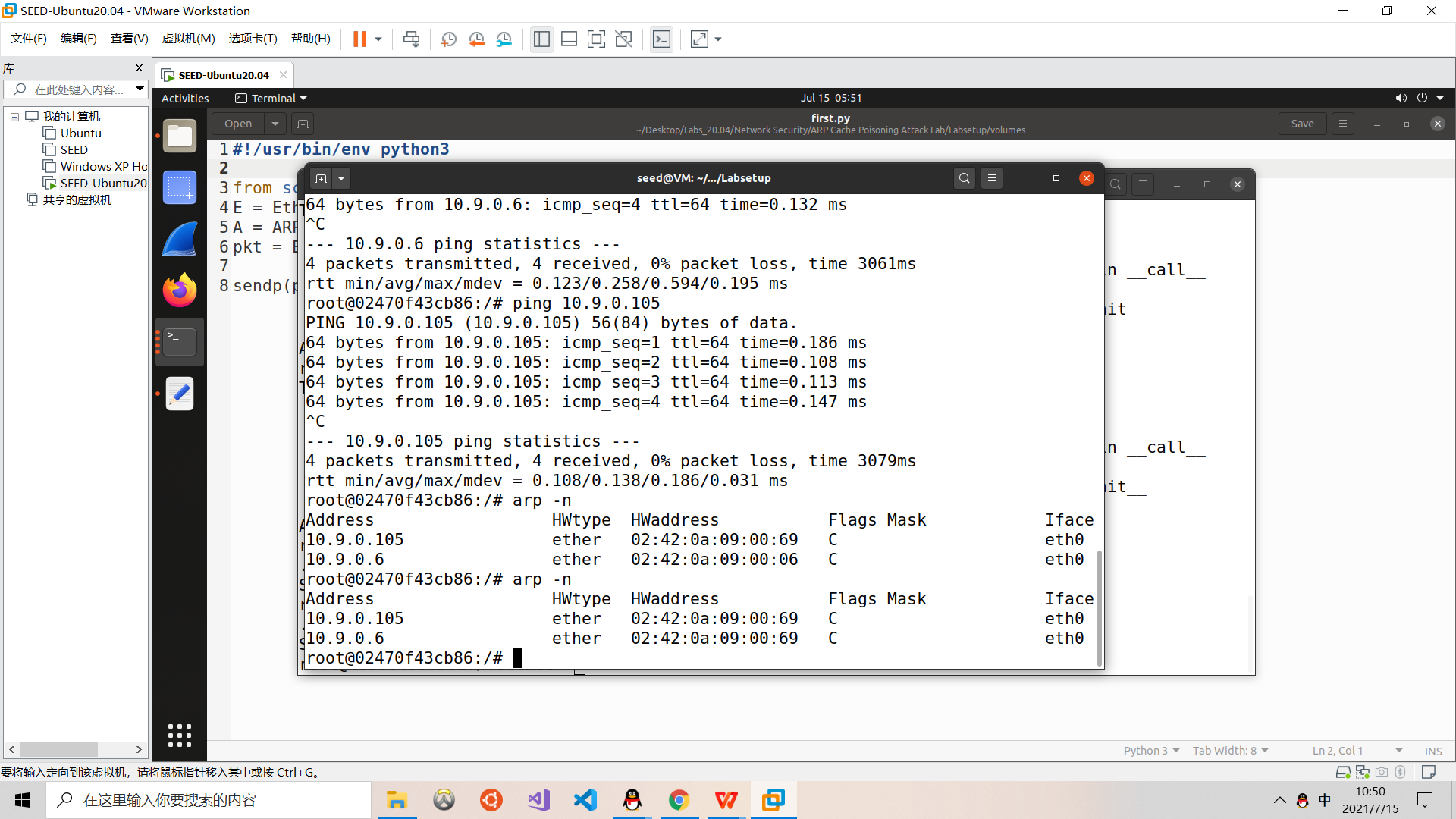
将脚本稍作修改，改为如下内容：



如下图所示，在arp缓存关于B的信息为空的情况下，脚本运行后并没有将B的IP地址映射到M的MAC地址，即arp request报文并不能起到ARP Cache Poisoning的作用。

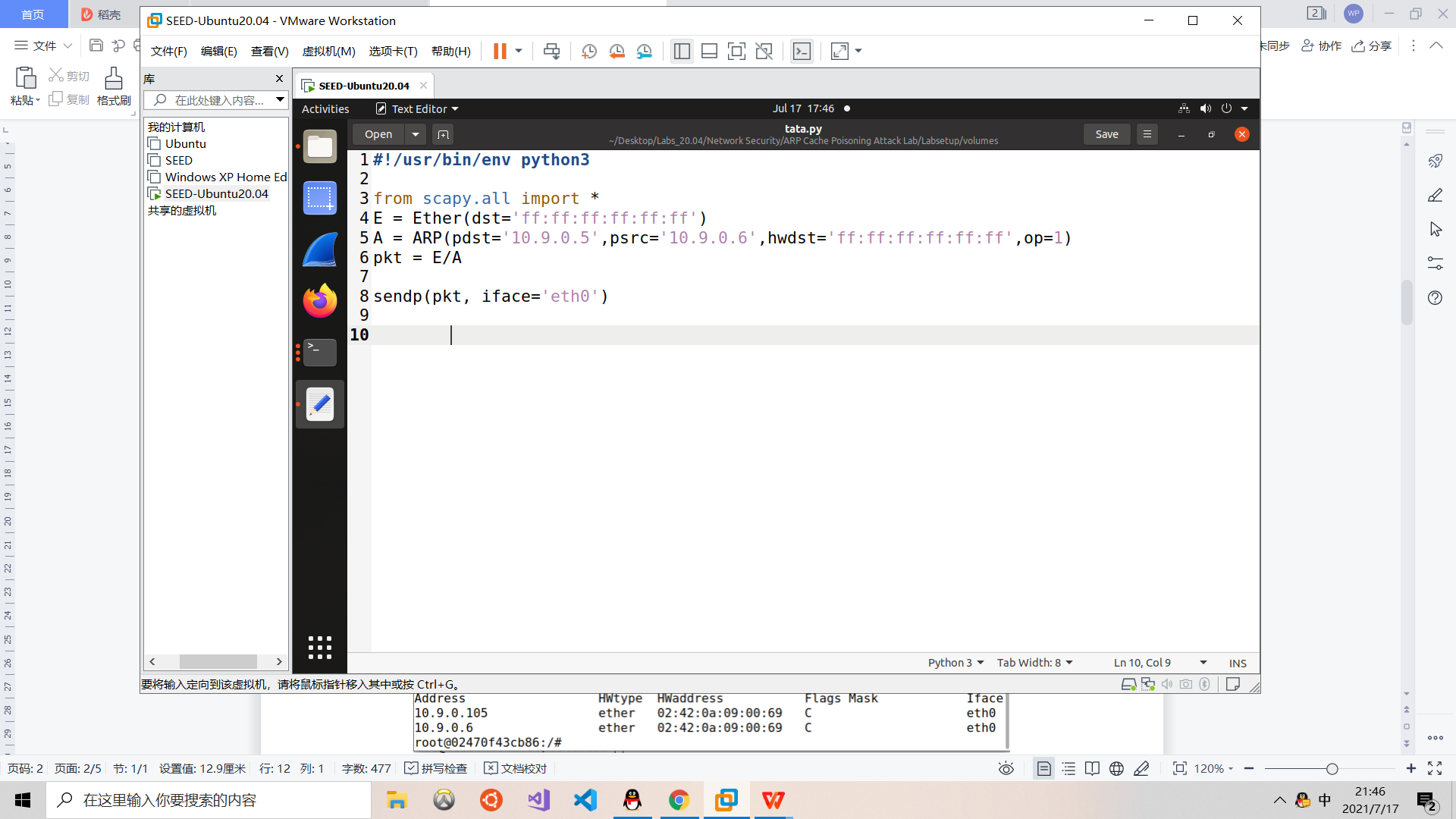


如图下所示，在在arp缓存关于B的信息不为空的情况下，脚本运行后成功将B的IP地址映射到M的MAC地址，起到了ARP Cache Poisoning的作用。

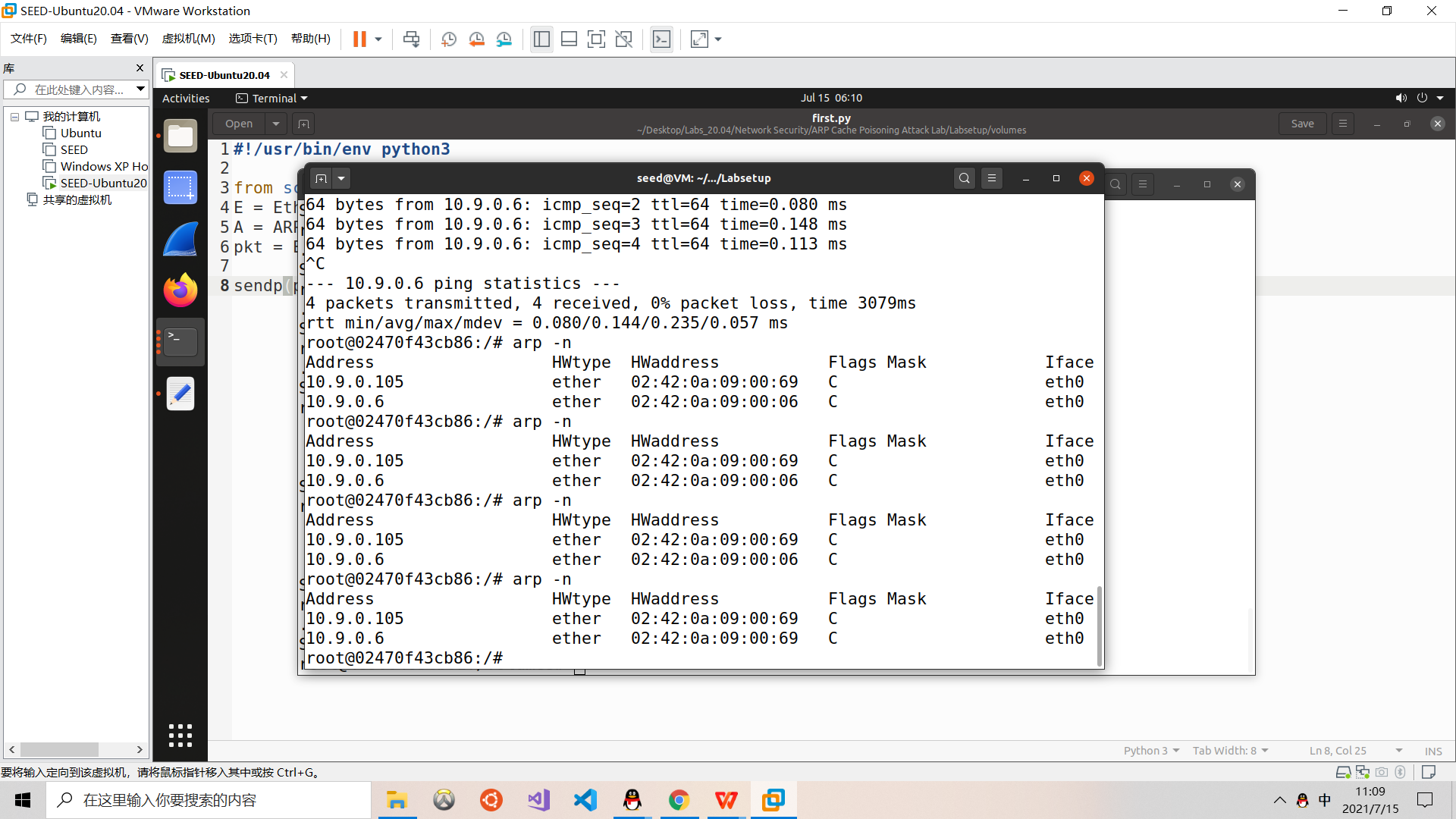


1. task1.C

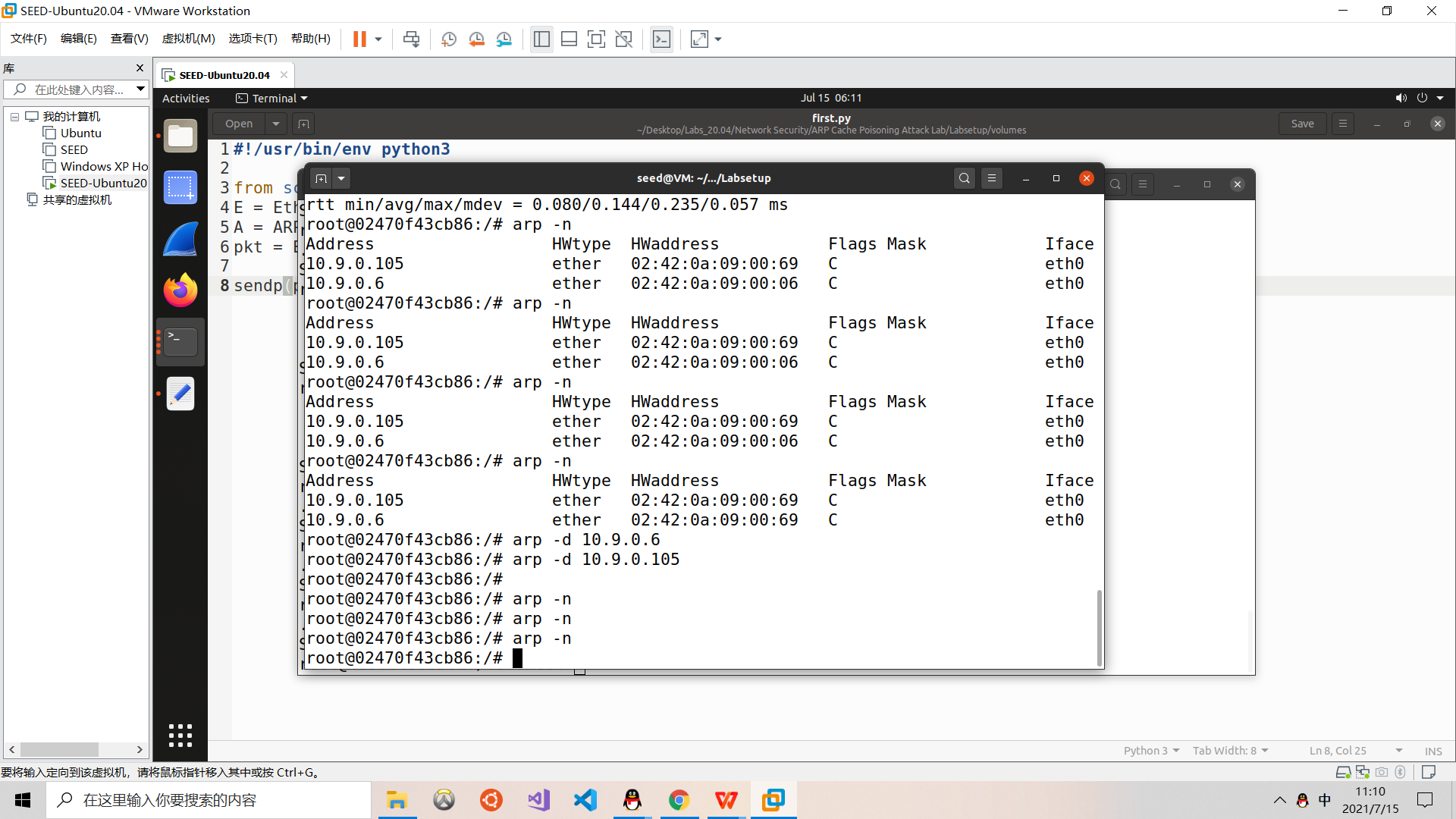
将脚本稍作修改，改为如下内容：



如下图所示，在arp缓存关于B的信息不为空的情况下，脚本运行后成功将B的IP地址映射到M的MAC地址，即免费arp（gratuitous ARP）报文起到了ARP Cache Poisoning的作用。



如下图所示，在arp缓存关于B的信息为空的情况下，脚本运行后并没有将B的IP地址映射到M的MAC地址，即免费arp（gratuitous ARP）报文并不能起到ARP Cache Poisoning的作用。

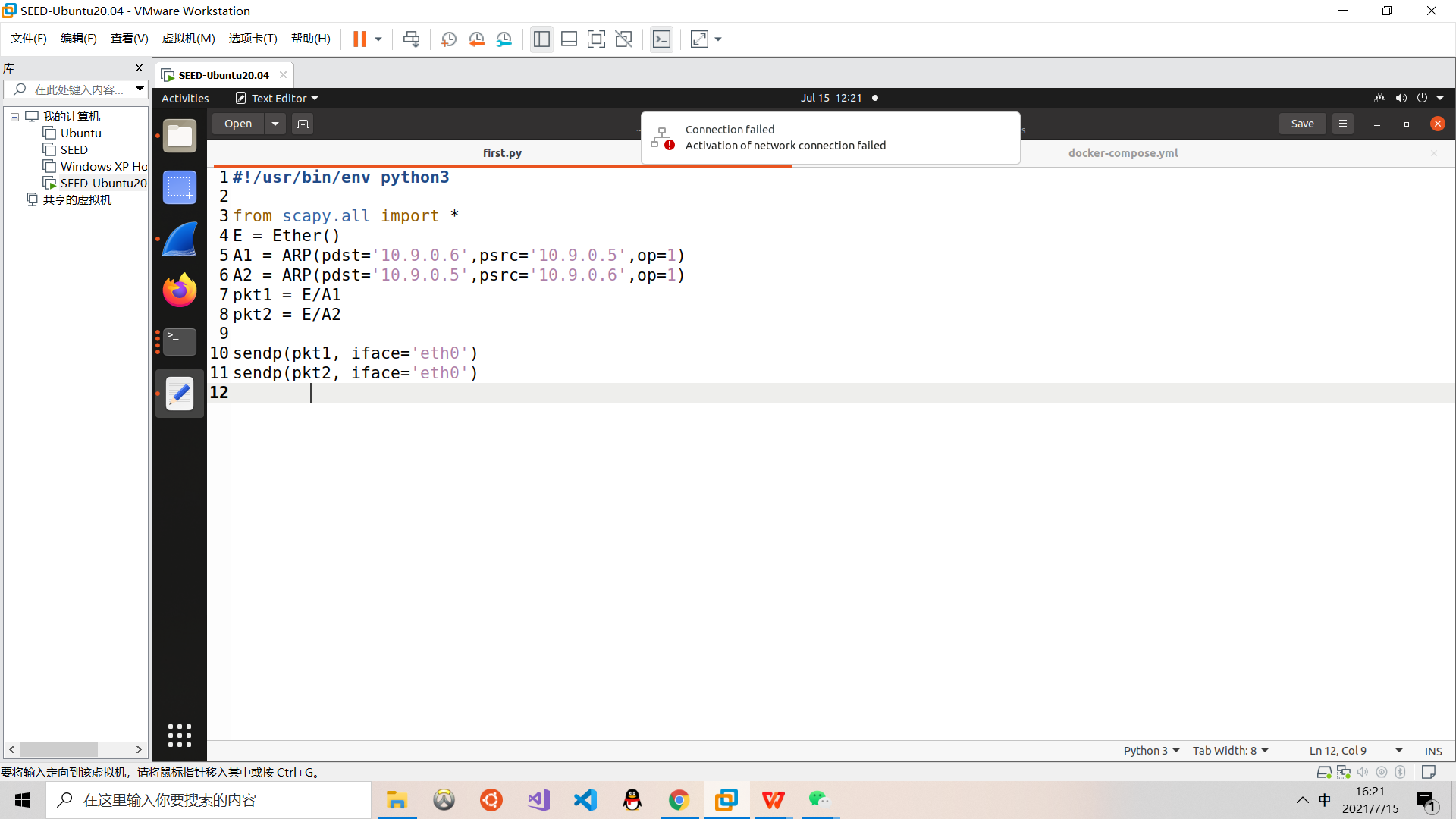


**Task 2: MITM Attack on Telnet using ARP Cache Poisoning**

**1）step 1**

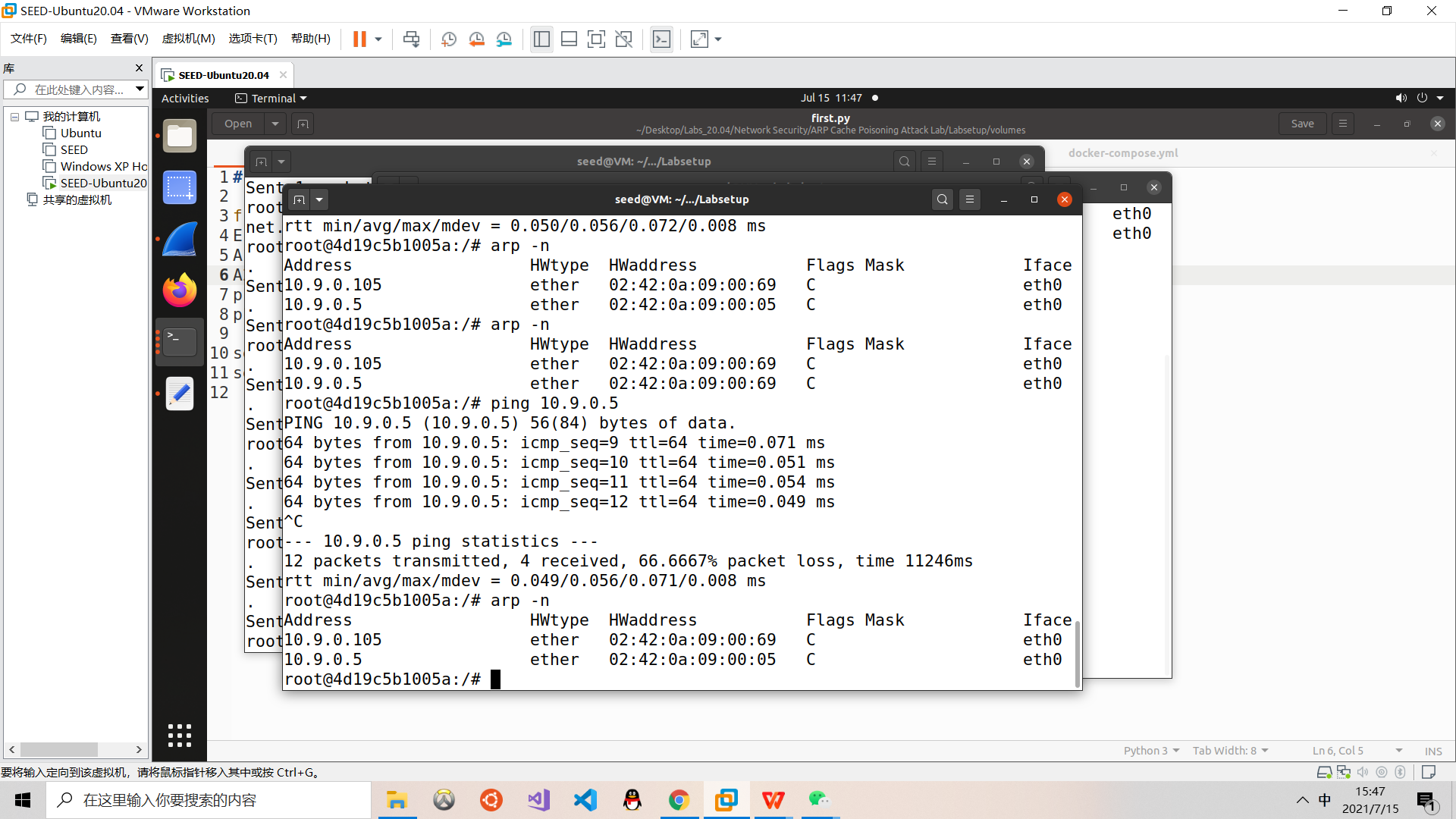
为了便于观察，我们在后台运行wireshark抓包。

首先，在10.9.0.105中运行如下脚本，该脚本作用是向A和B同时发起ARP缓存投毒攻击：



**2）step 2**

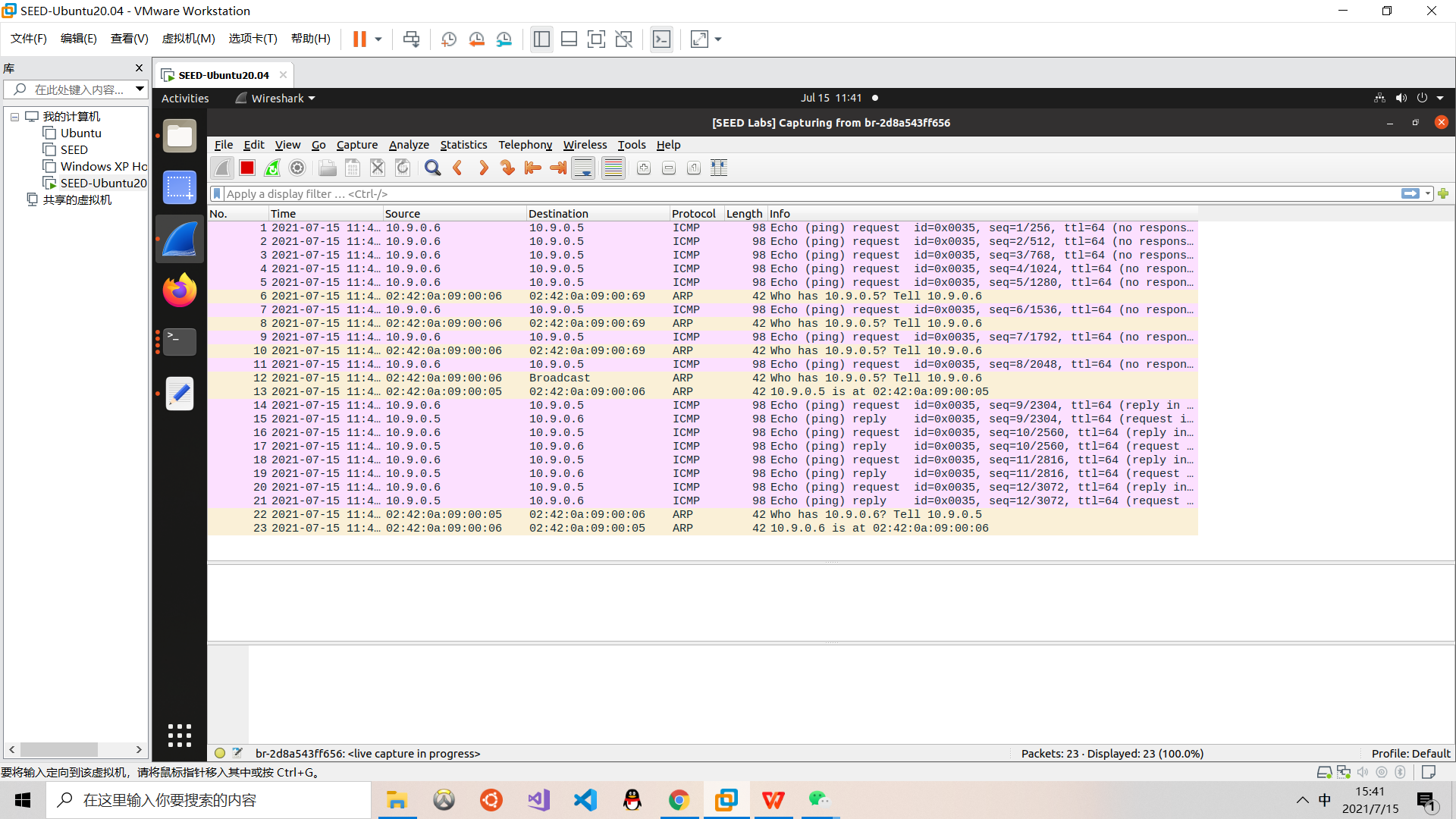
先在M上输入命令sysctl net.ipv4.ip\_forward=0，然后进入B 10.9.0.6（进入A或B是一样的），如果输入arp -n命令，会看到此时arp缓存已经成功写入，A的IP地址映射到了M的MAC地址。尝试ping 10.9.0.5，会出现先ping不通，但过数秒钟后就ping通。我们结束ping，可以看到是有部分包丢失的（也就是之前ping不通的部分）。这时候再输入arp -n，发现arp缓存的内容被更改了，A的IP地址正确地映射到了自身的MAC地址。



可以通过wireshark抓到的包可以看到由于自动转发被关闭，前5次ping都没有收到回复。

这时B就会自动先向M单播一个arp请求报文，寻找10.9.0.5对应的MAC，由于M的IP地址不是10.9.0.6，而我们也没有构造响应spoof包，所以无法回应。

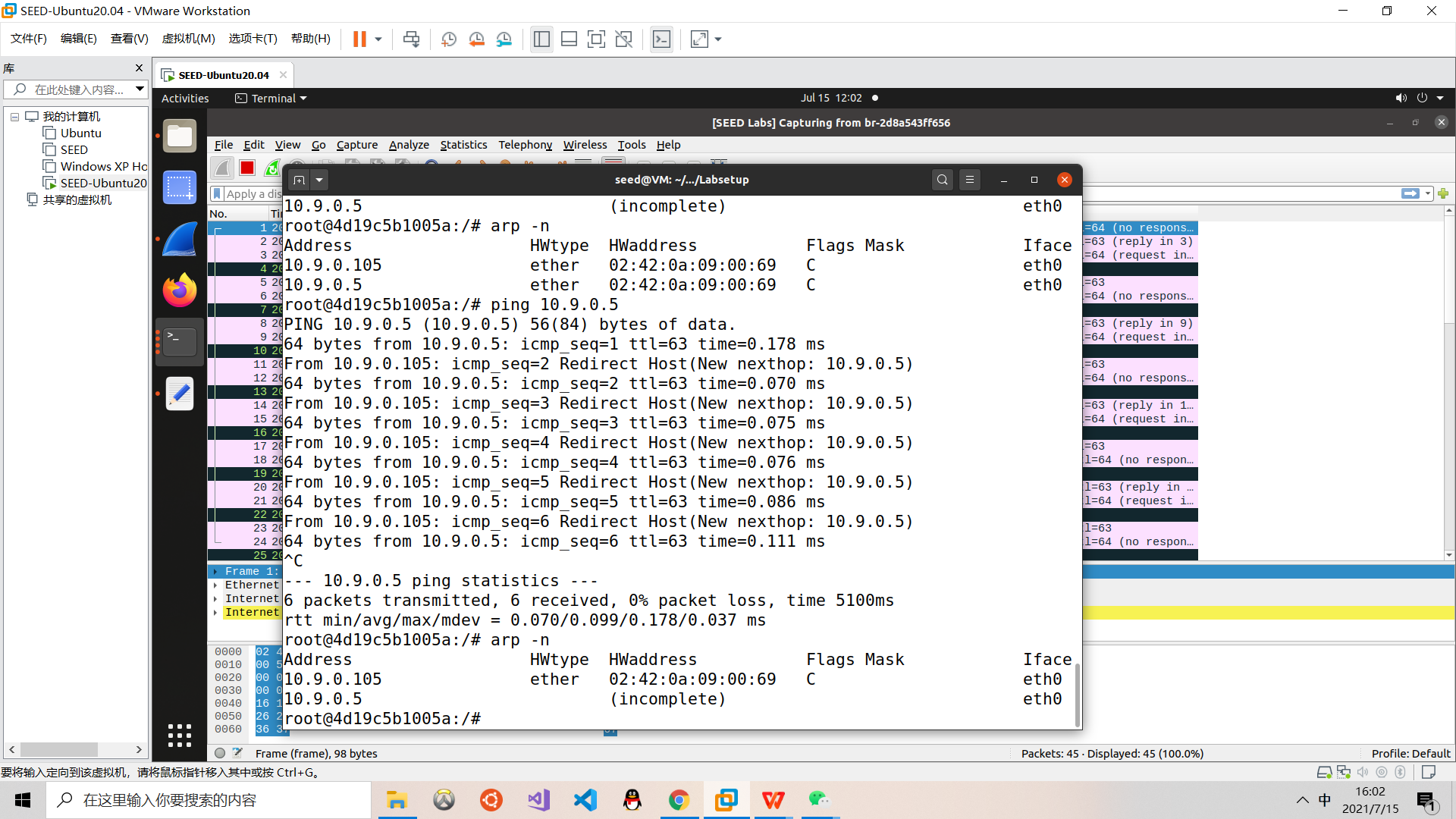
在向M单播3次收不到回应后，B就会自动广播一个arp请求报文，寻找10.9.0.5对应的MAC，之后便是A给予回应，二者也成功建立连接。



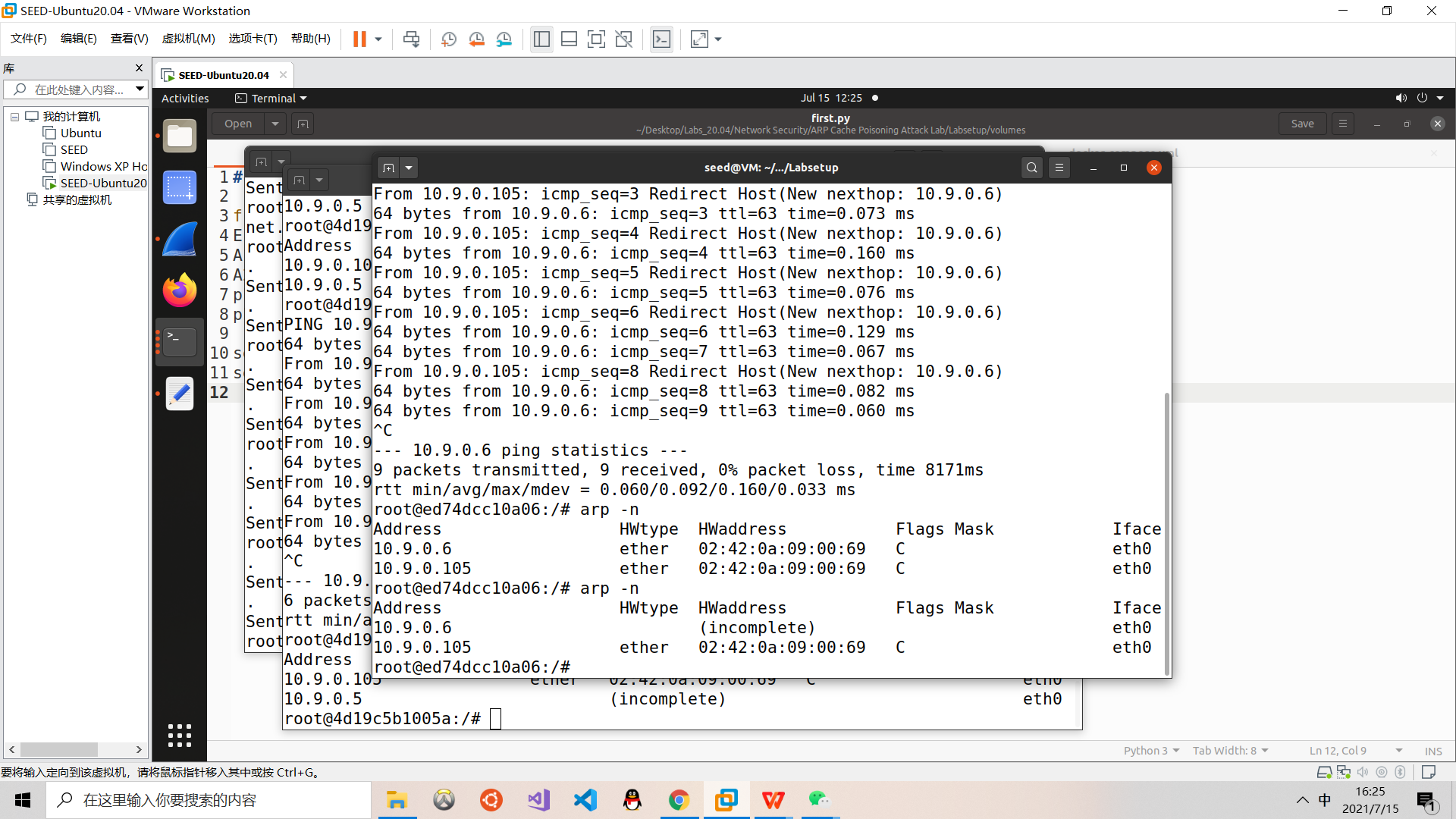
**3）step 3**

先在M上输入命令sysctl net.ipv4.ip\_forward=1，打开IPv4报文自动转发。然后进入B 10.9.0.6（进入A或B是一样的），如果输入arp -n命令，会看到此时arp缓存已经成功写入，A的IP地址映射到了M的MAC地址。尝试ping 10.9.0.5，每次都是可以ping通的，无数据报未送达的情况。但是每次ping过后，都会收到一个来自M的重定向的报文。

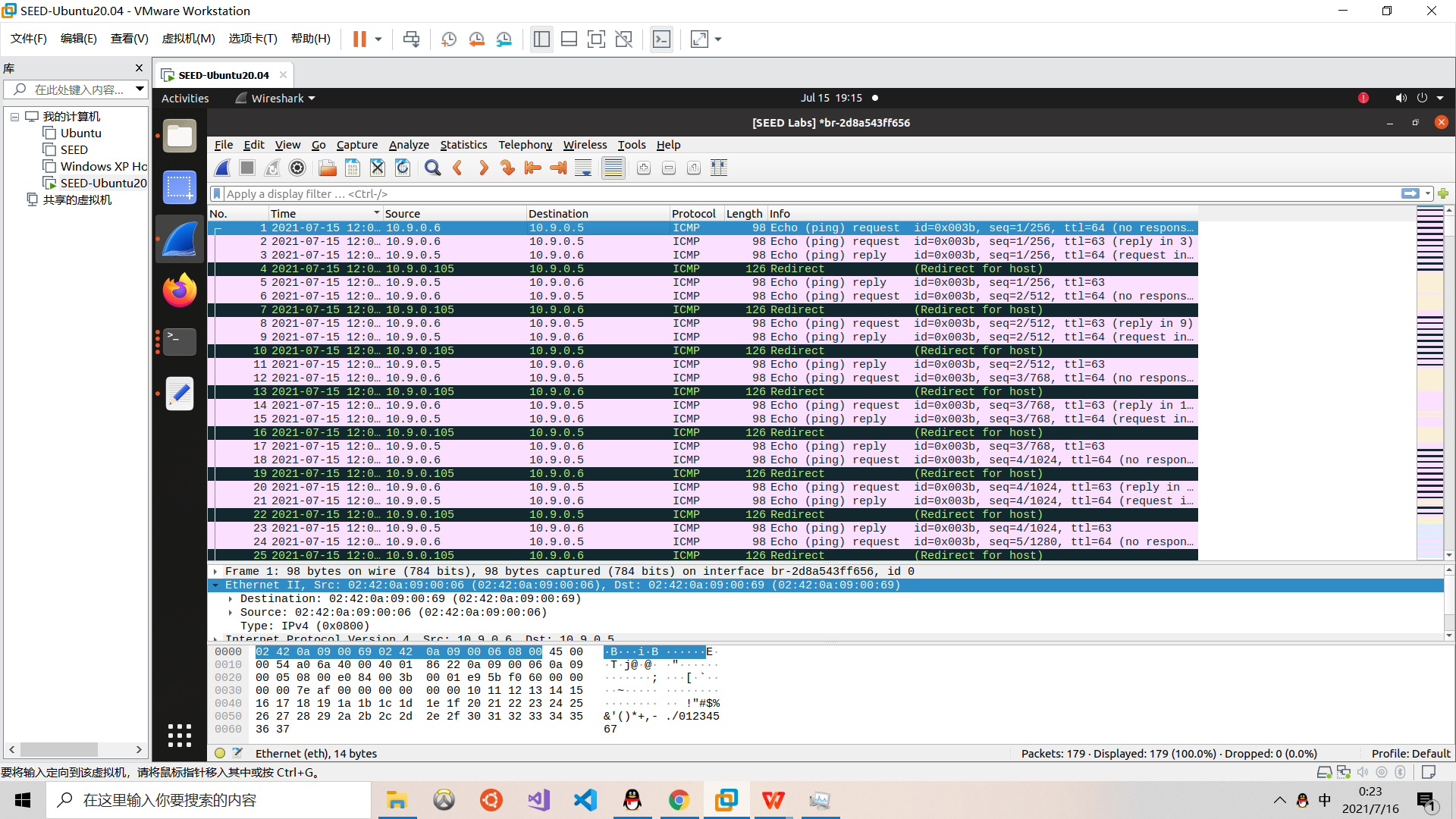
ping结束后，arp缓存变化，A对应的MAC地址变成了 (imcomplete)。



这时后A中，B对应的MAC地址也变成了 (imcomplete)。

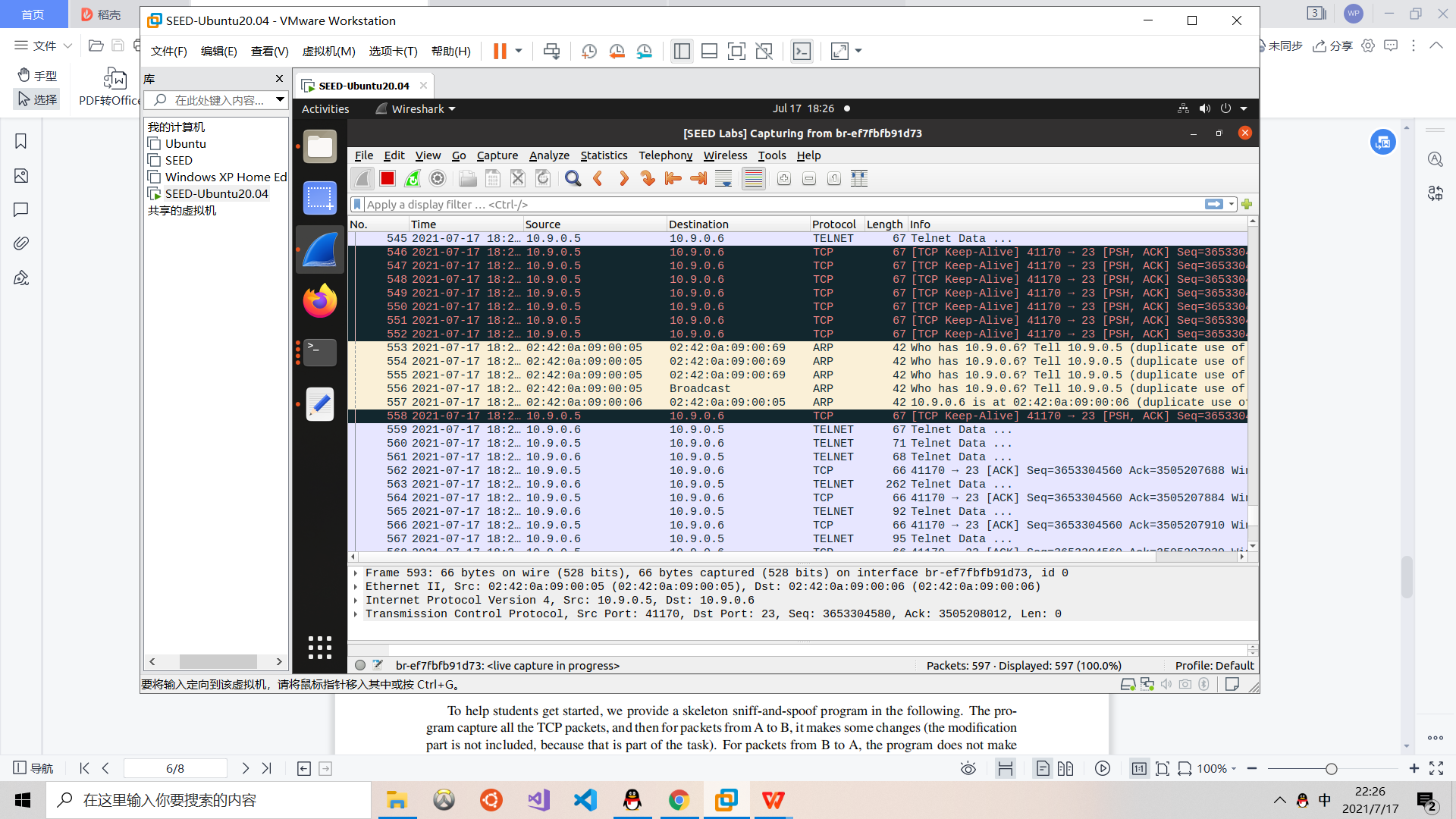


从wireshark抓包结果也能看出来，M确实起到了中间人的转发作用，A、B之间的通信都经过了M。但是在每次M转发之后，都会对A/B发送一个ICMP重定向报文，告知A/B下次通信不必再发给自己，可以直接发给对方（因为在同一局域网内可以直接送达），这是因为计算机底层并不知道我们对A、B进行了ARP缓存攻击而企图纠正A、B的错误。但是ICMP作为网络层协议，给出的重定向自然是IP地址重定向，但中间人M的出现是由于A/B的IP地址映射到了M的MAC地址产生的，实际上是MAC地址错误导致，所以ICMP重定向报文不会起到作用。

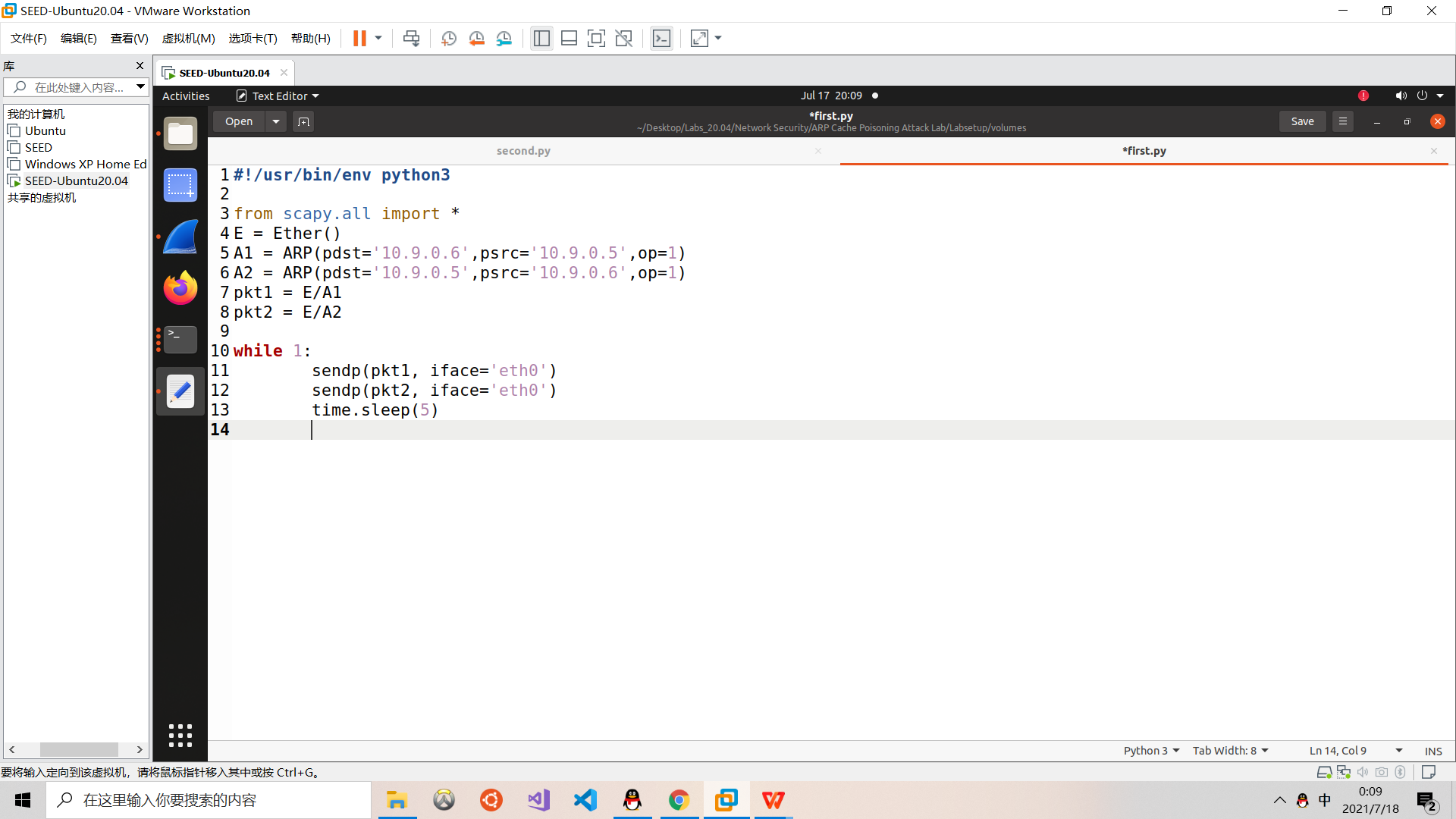


**4）step 4**

运行step 2里的脚本，然后进入A，输入命令telnet 10.9.0.6。成功建立连接后，关闭M上的IPv4自动转发，尝试在A终端上输入命令，此时没有任何回显。但过数秒钟后命令就会出现。查看wireshark，可以发发现情况和step2是类似的。A就会自动先向M单播一个arp请求报文，而M无法回应。在向M单播3次收不到回应后，A就会自动广播一个arp请求报文，寻找10.9.0.6对应的MAC，之后便是B给予回应，二者也成功建立连接。



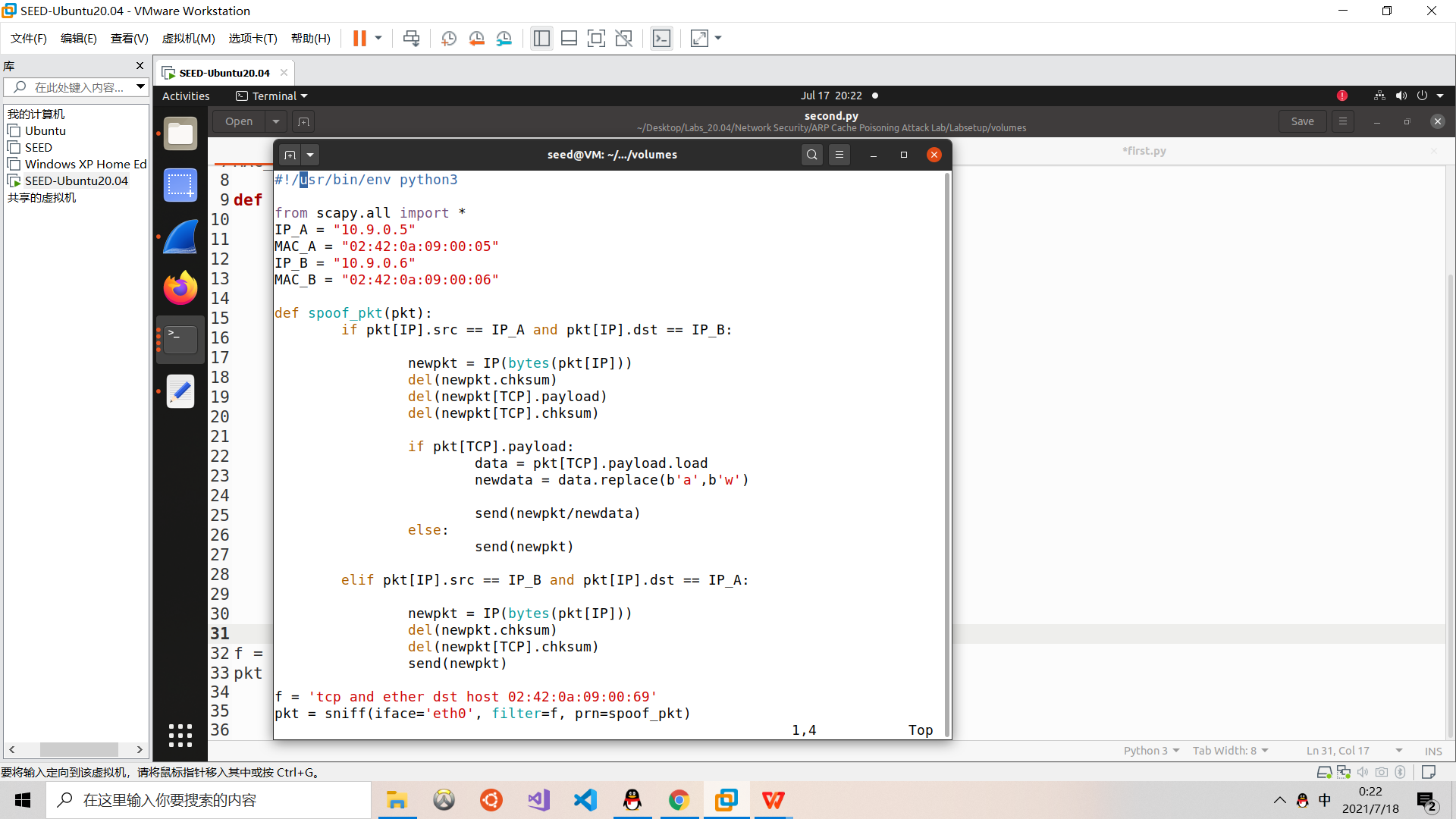
我们对脚本稍作修改，为了避免arp缓存过期。我们设置为每隔5s发一次，也就是说该脚本会持续运行。



首先我们在10.9.0.105上输入命令sysctl net.ipv4.ip\_forward=1，确保此时自动转发IPv4数据报的功能是开启的，然后运行上面修改过的可持续运行的脚本进行arp投毒。

在10.9.0.5上输入命令telnet 10.9.0.6，用户名seed，密码dees。

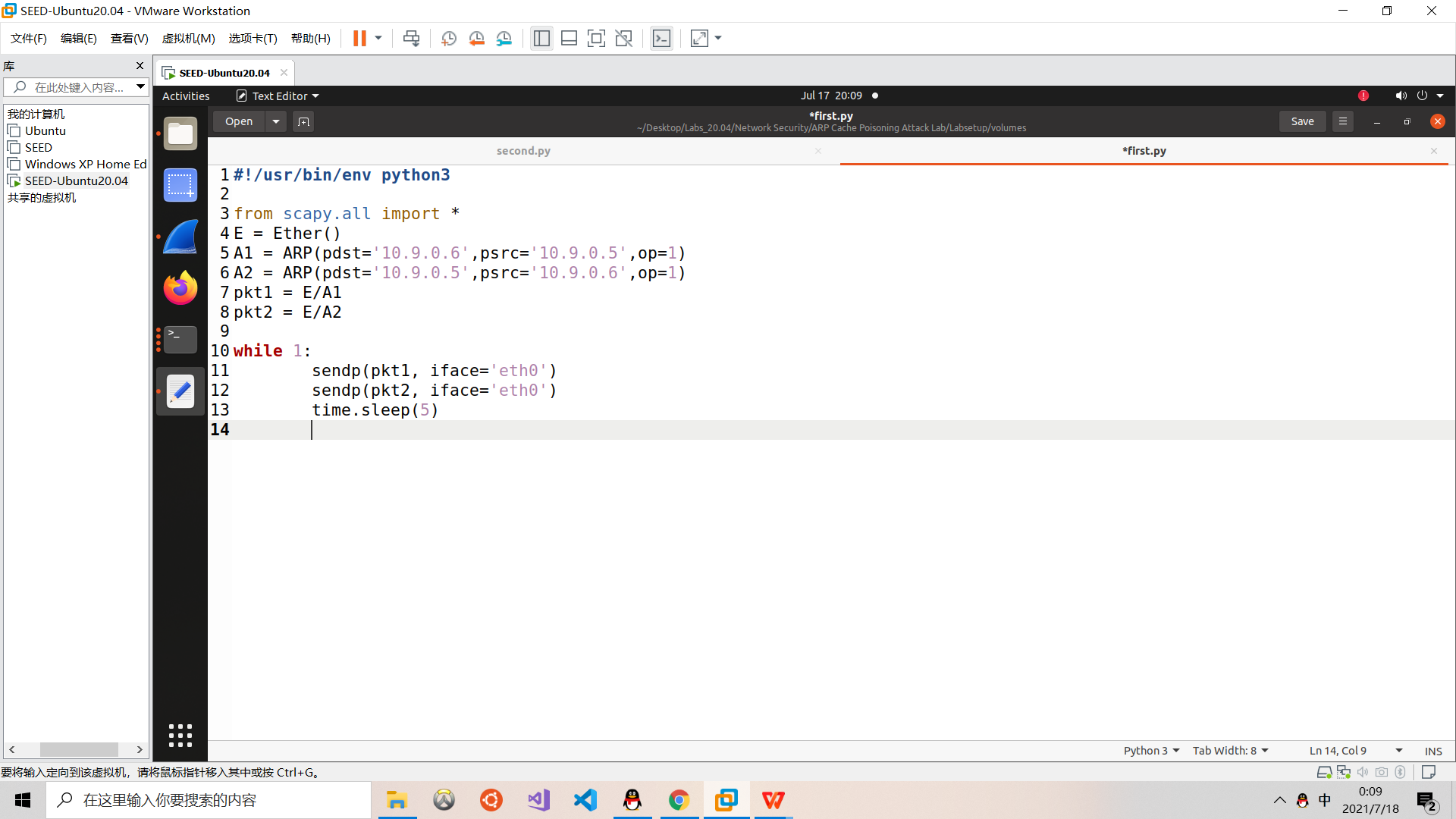
成功建立连接后。再打开一个10.9.0.105的docker，输入命令sysctl net.ipv4.ip\_forward=0，关闭自动转发IPv4数据报的功能，然后运行如下脚本：



该脚本的功能是，在劫持后将输入的字符串“a”替换为“w”。由于telnet的通信方式是A（客户端）每键入一个字符，该字符都会作为数据打包成一个TCP报文发给B（服务器），B将该字符收入缓存后再把字符回传给A，直到收到回车字符后才会解析成一个完整命令。所以在A终端上看到的字符其实是B收到A的消息回传给A的，在本例中由于我们劫持更改，就会表现为，输入“a”却显示的是“w”。

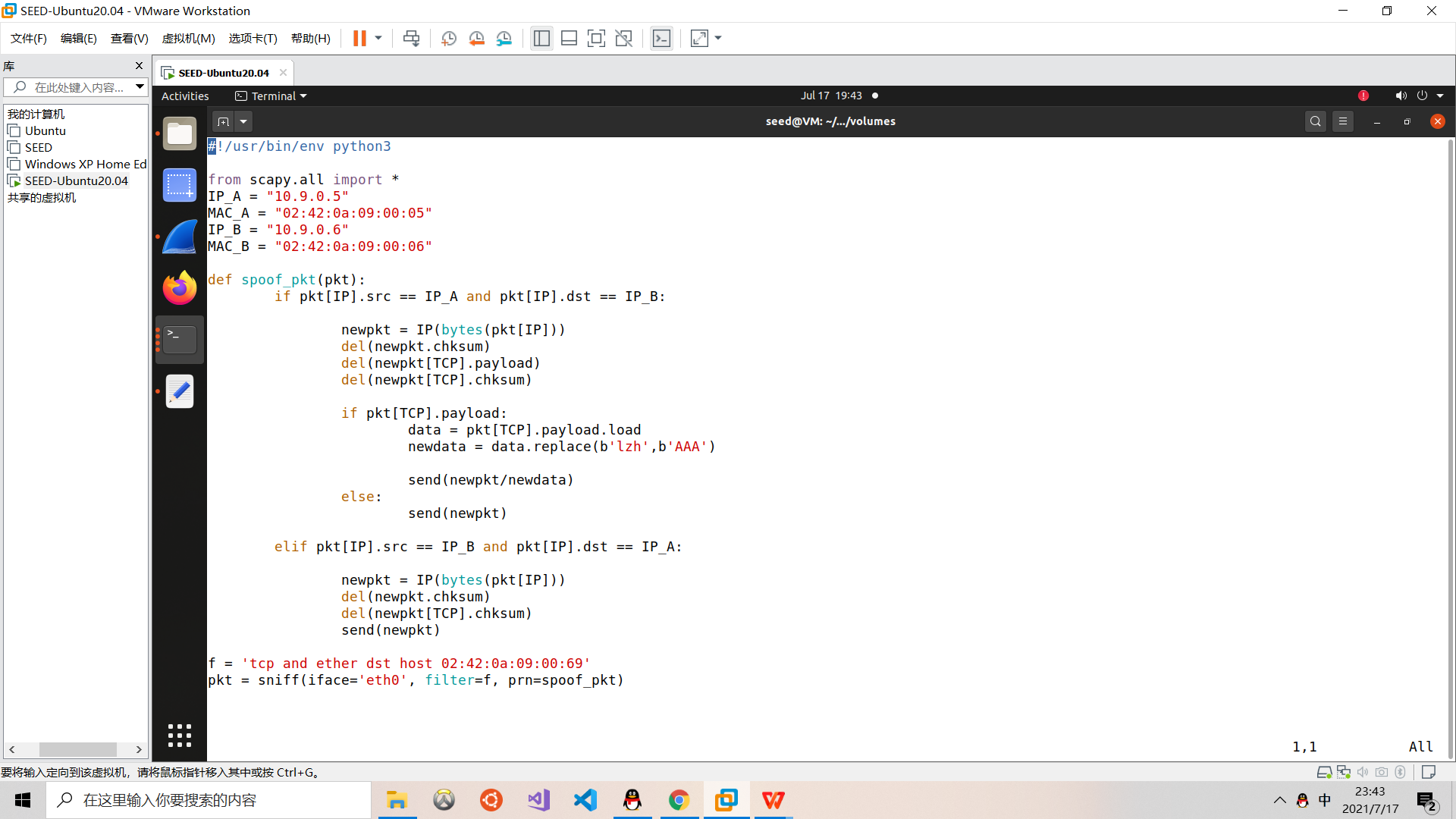
**Task 3: MITM Attack on Netcat using ARP Cache Poisoning**

步骤与上一实验相似，首先我们在10.9.0.105上输入命令sysctl net.ipv4.ip\_forward=1，确保此时自动转发IPv4数据报的功能是开启的，然后运行如下脚本，进行ARP缓存投毒攻击：



在10.9.0.6和10.9.0.5上分别输入命令nc -lp 9090和nc 10.9.0.6 9090，建立netcat连接。

成功建立连接后。再打开一个10.9.0.5的docker，在10.9.0.105输入命令sysctl net.ipv4.ip\_forward=0，关闭自动转发IPv4数据报的功能，然后运行如下脚本：



该脚本的功能是，在劫持后将输入的字符串“lzh”替换为“AAA”。

结果如下：

