**第五次实验**

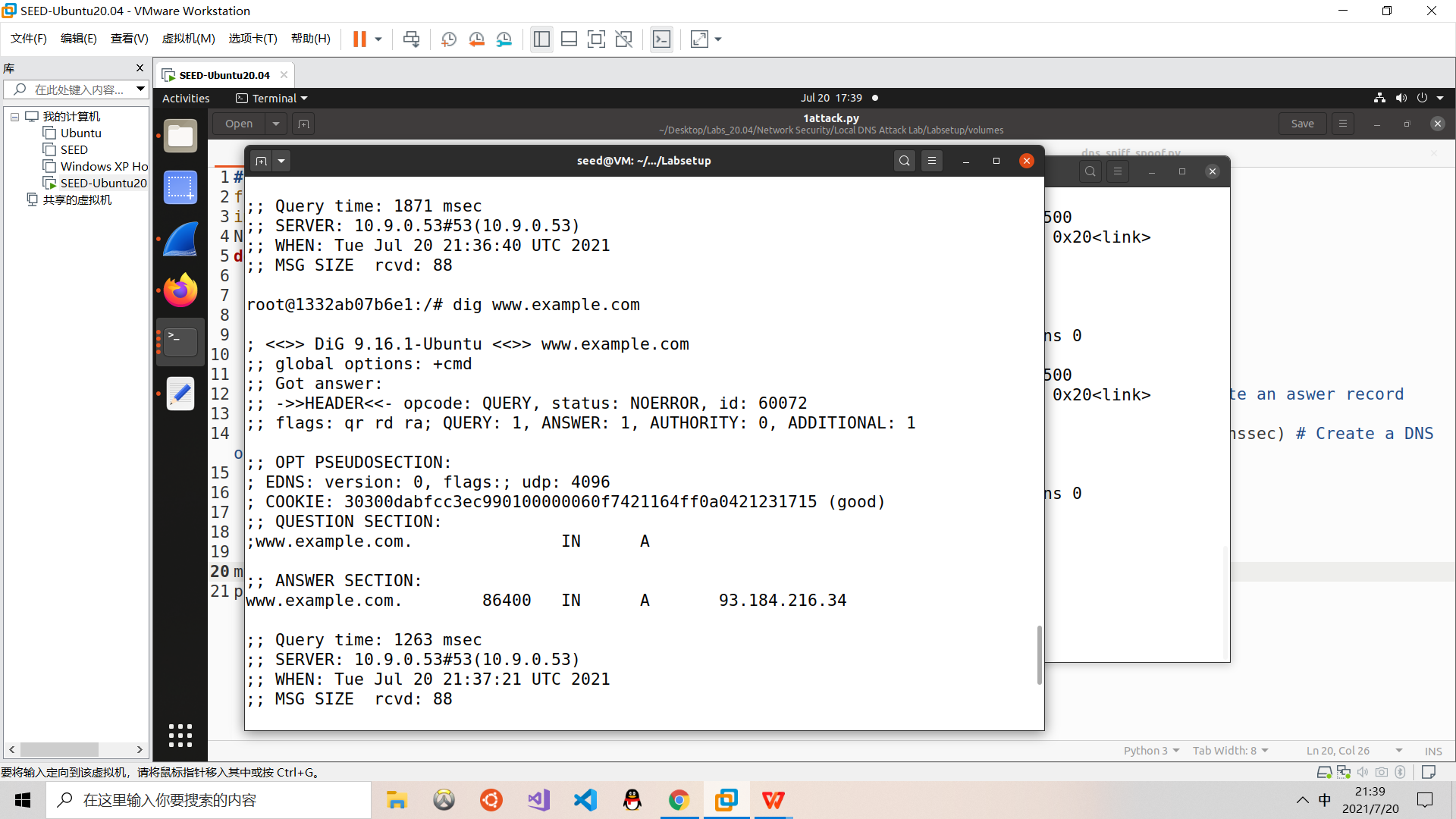
57118122 李泽厚

**Task 1: Directly Spoofing Response to User**

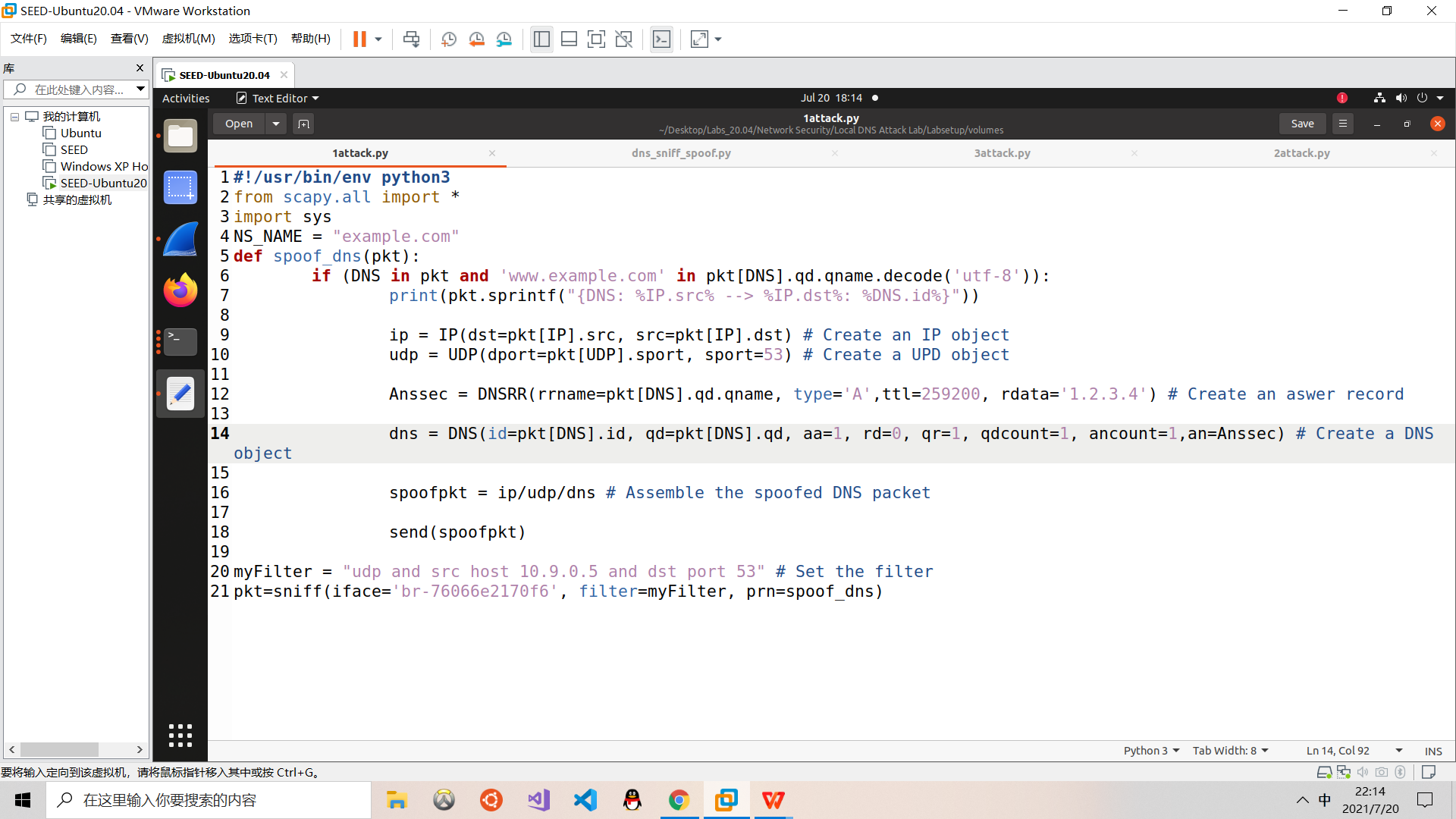
在本实验中，我们对单个主机10.9.0.5进行攻击。

为了使伪造回复比合法回复传回的速度更快，我们在本地DNS服务器10.9.0.53上输入命令 tc qdisc add dev eth0 root netem delay 200ms（保持此配置在之后的实验中一直生效），增加延迟200ms。然后输入 rndc flush，刷新本地DNS服务器缓存。

在受害者机器上输入命令 dig www.example.com，查看攻击前的结果，如下图：



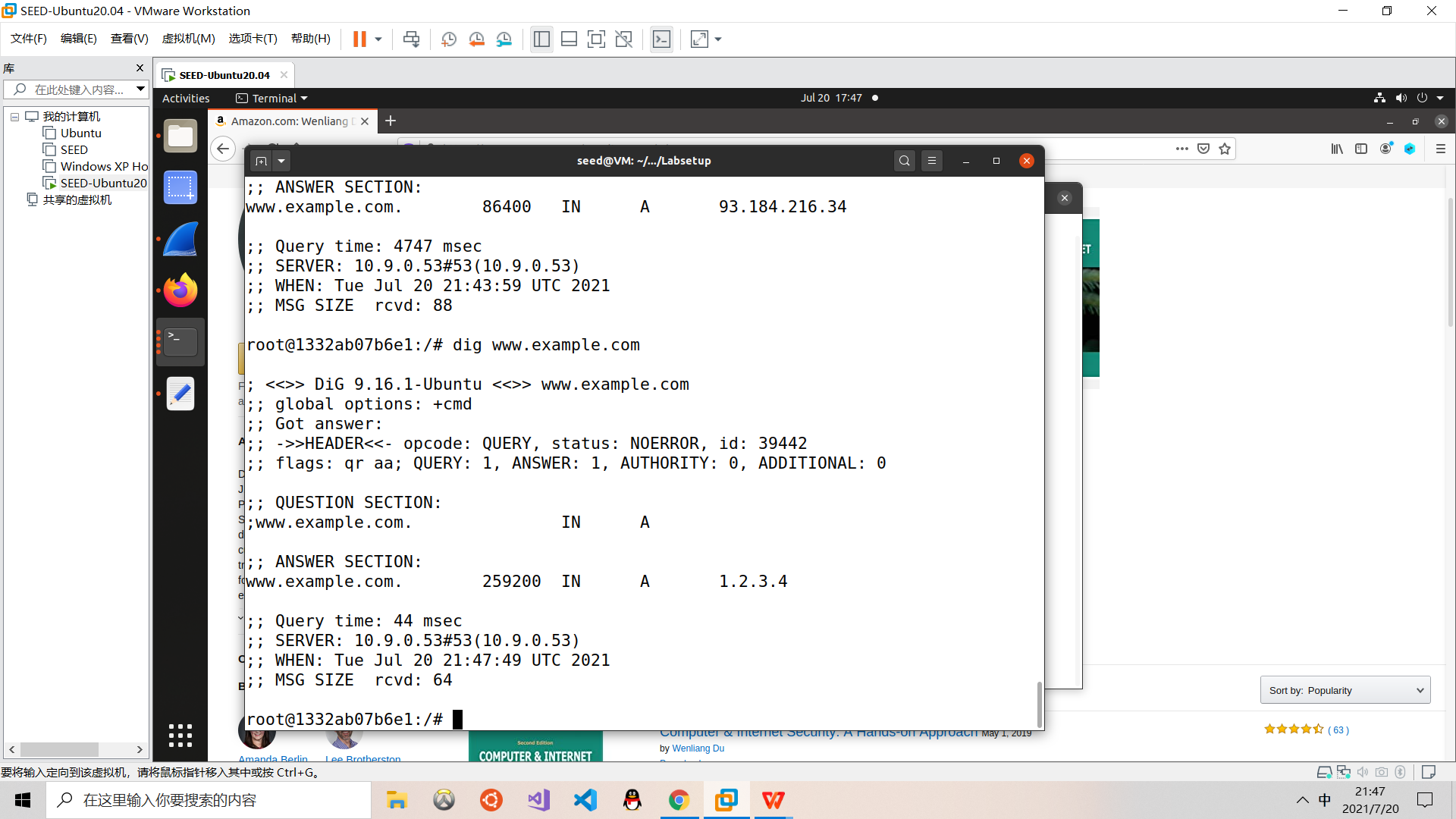
然后在攻击者主机10.9.0.1上运行如下Python程序：



该脚本中，我们对受害者主机发起攻击，让受害者把www.example.com的ip地址解析为1.2.3.4。

本地DNS服务器上输入 rndc flush，刷新缓存。

再次在受害者机器上输入命令 dig www.example.com，得到如下结果：

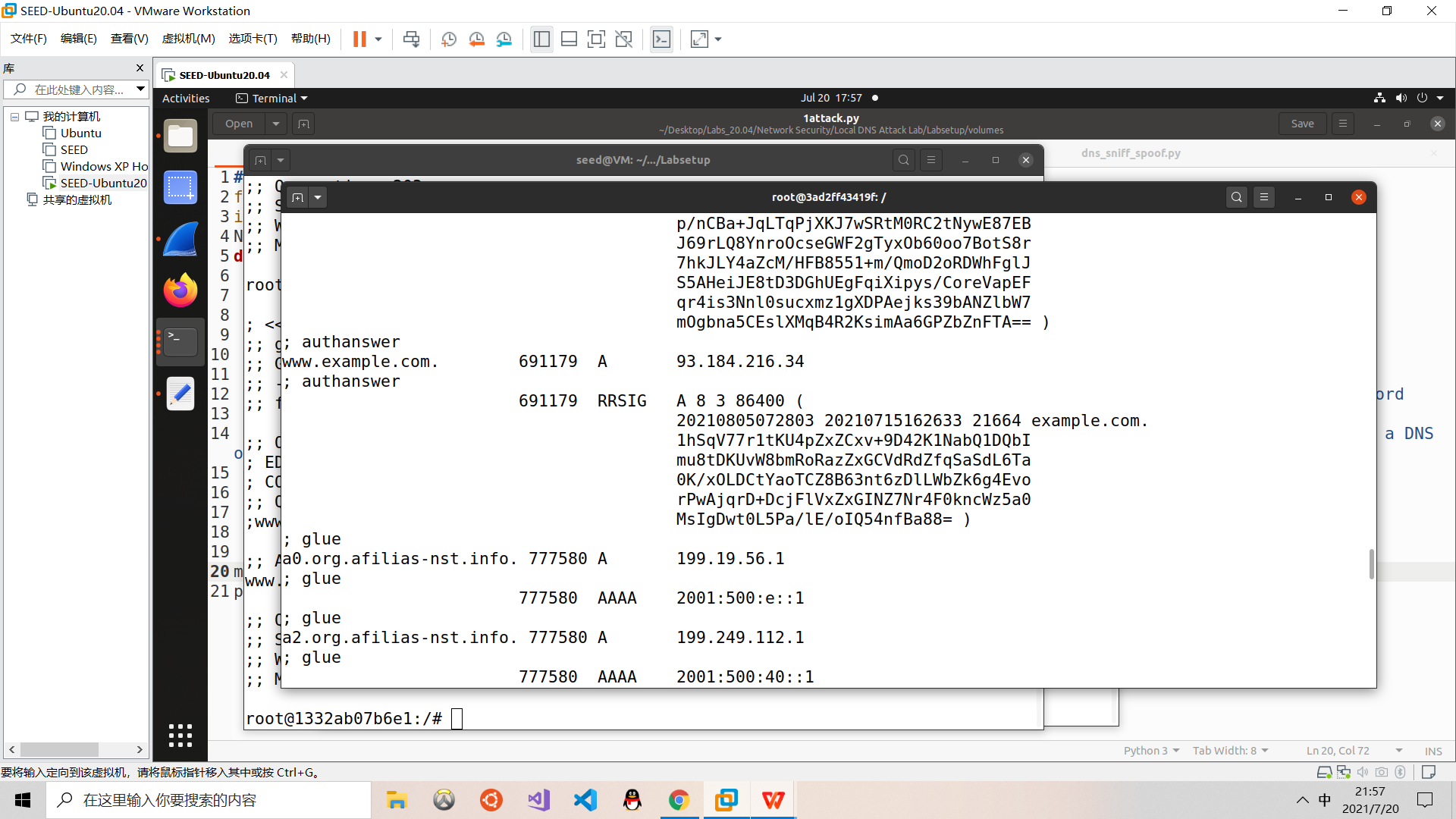


从图中可以看出，受害者错误地将www.example.com的ip地址解析为1.2.3.4。

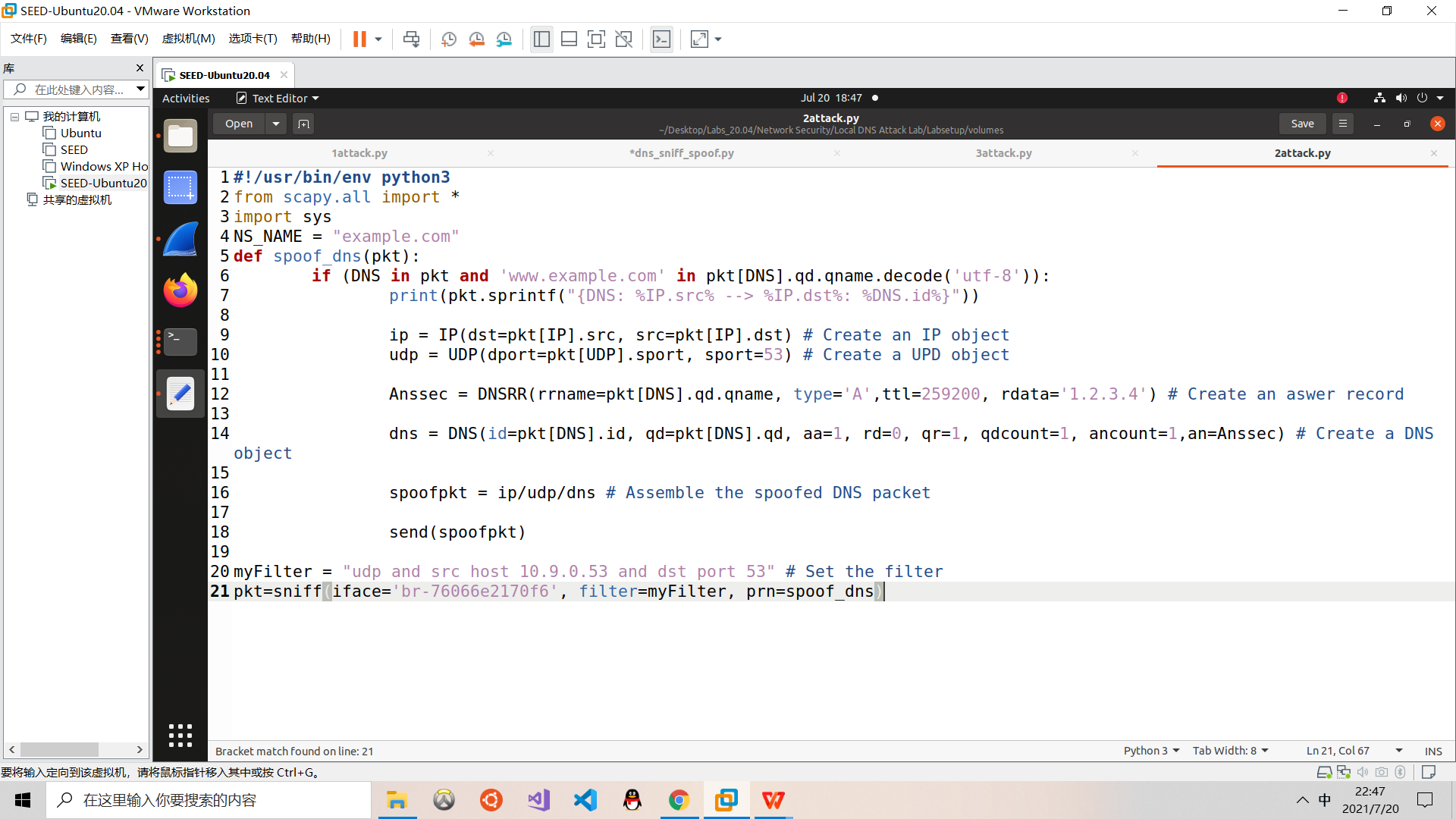
**Task 2: DNS Cache Poisoning Attack – Spoofing Answers**

在本实验中，我们对本地DNS服务器10.9.0.53进行攻击。

与前面类似。先在本地DNS服务器10.9.0.53上输入命令 rndc flush刷新缓存，然后在受害者机器上输入命令 dig www.example.com。再进入10.9.0.53，输入 rndc dumpdb -cache将缓存导入一个文件中，输入cat /var/cache/bind/dump.db查看该文件，我们仔细翻找，可以找到如下图内容：



然后在攻击者主机10.9.0.1上运行如下Python程序（task1脚本稍作修改）：



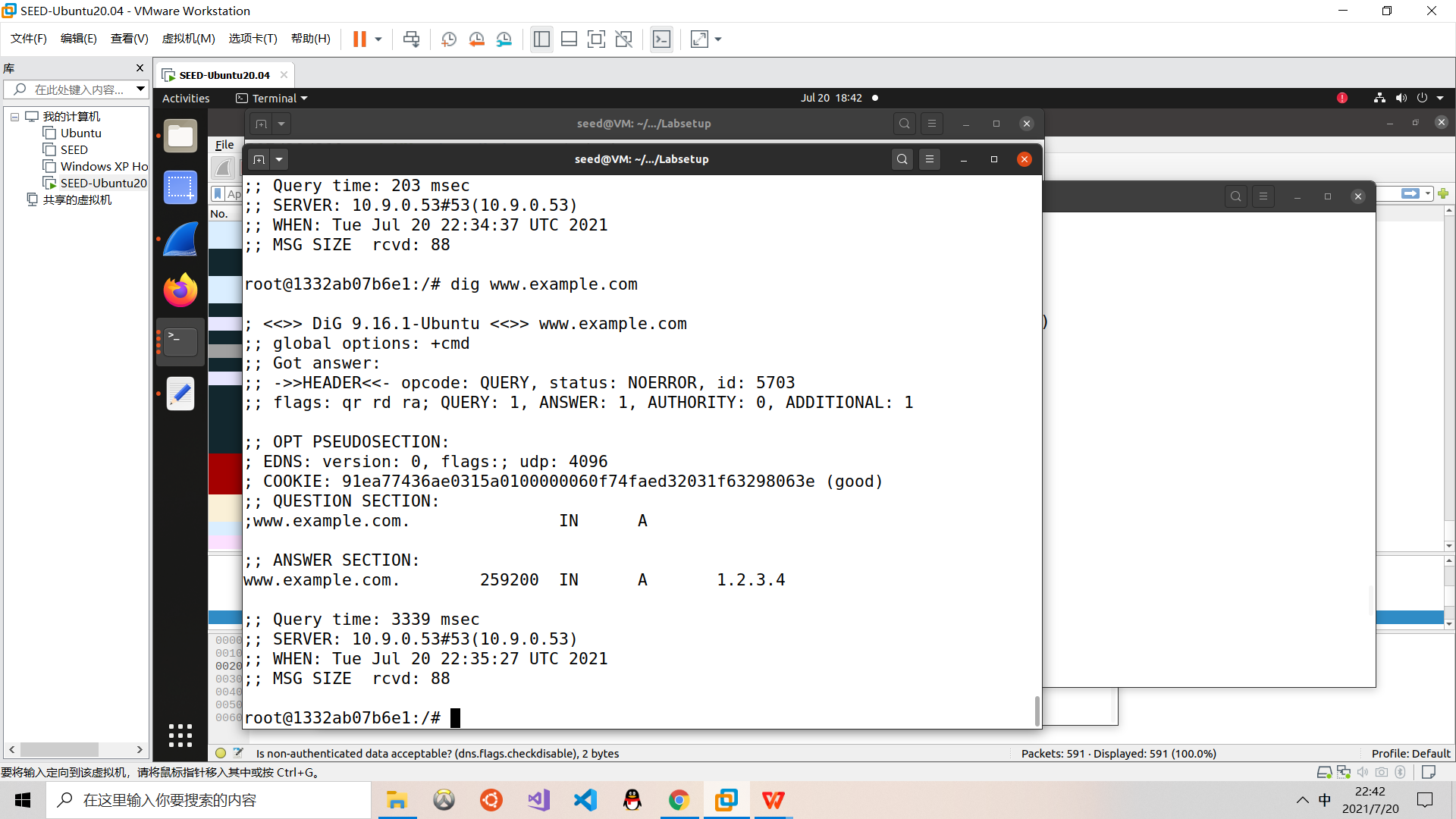
该脚本中，我们对本地DNS服务器发起攻击，通过劫持对话，让本地DNS服务器把www.example.com的ip地址解析为1.2.3.4。

第一次实验时，脚本显示发包成功，但是无论DNS服务器还是客户端，都没显示攻击成功，猜测是伪造回复比合法回复传回的速度慢了。于是进入router，输入命令tc qdisc add dev eth0 root netem delay 300ms（保持此配置在之后的实验中一直生效），增加300ms通讯延迟，进一步减缓合法回复的速度。

运行脚本；

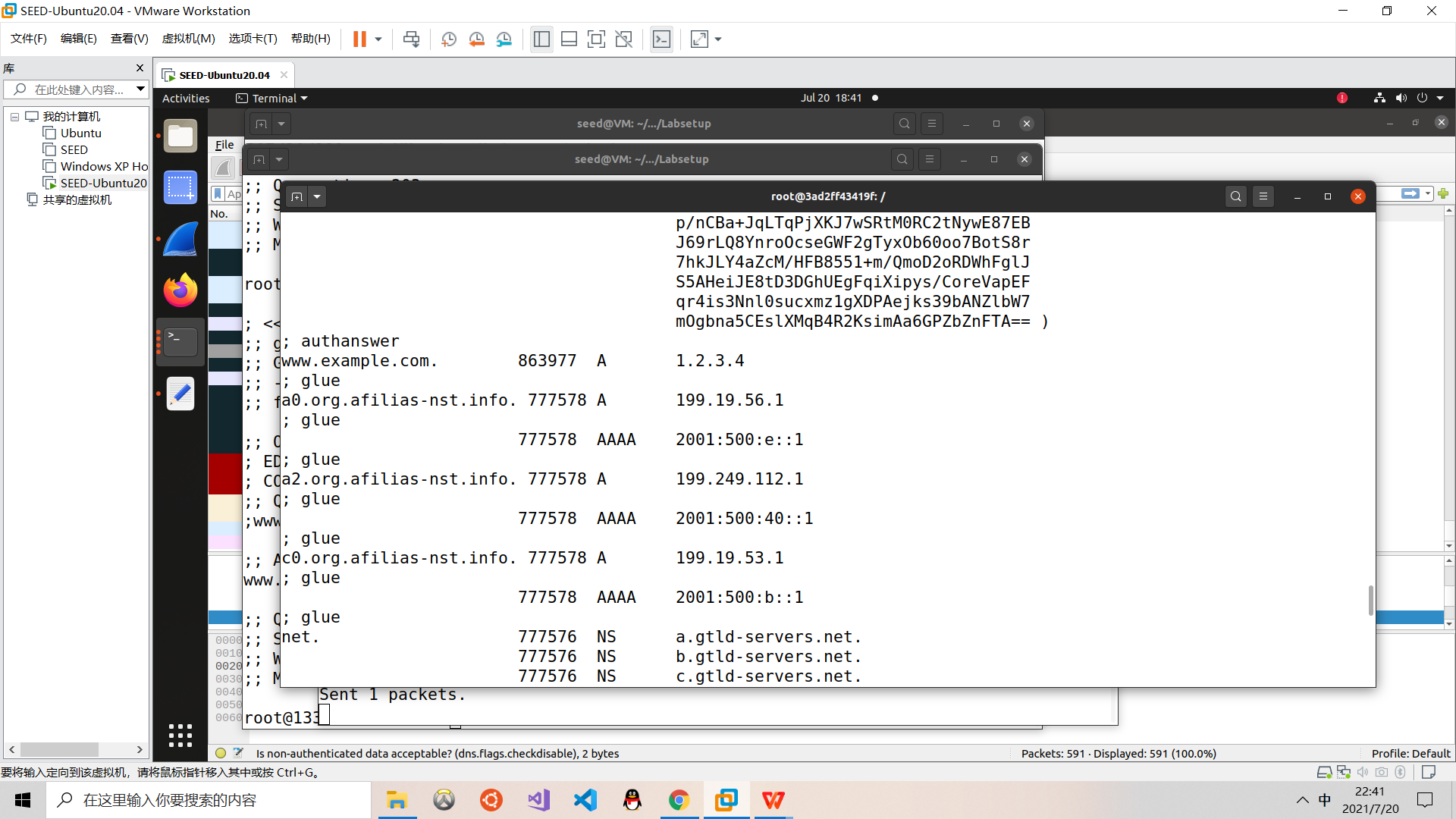
本地DNS服务器上输入 rndc flush，刷新缓存；

再次在受害者机器上输入命令 dig www.example.com，得到如下结果：



从图中可以看出，受害者错误地将www.example.com的ip地址解析为1.2.3.4。

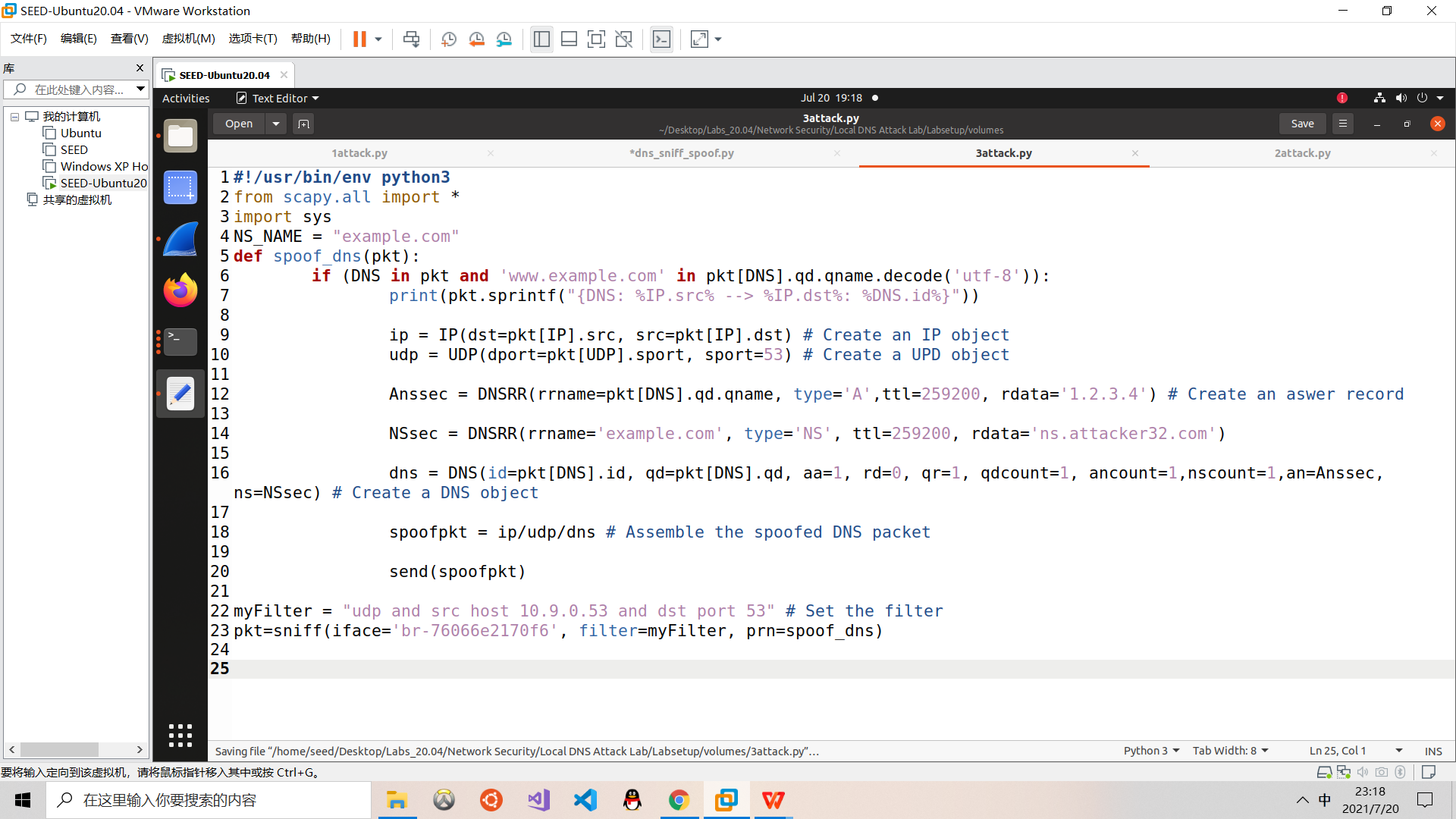
我们进入本地DNS服务器，先后输入命令rndc dumpdb -cache和cat /var/cache/bind/dump.db，在导出的缓存文件中可以找到如下结果：



可见在本地DNS服务器中，已经错误地将www.example.com的ip地址解析为1.2.3.4。这样一来该局域网内所有主机都会将www.example.com的ip地址解析为1.2.3.4。

**Task 3: Spoofing NS Records**

对task2中负载部分稍微增加：

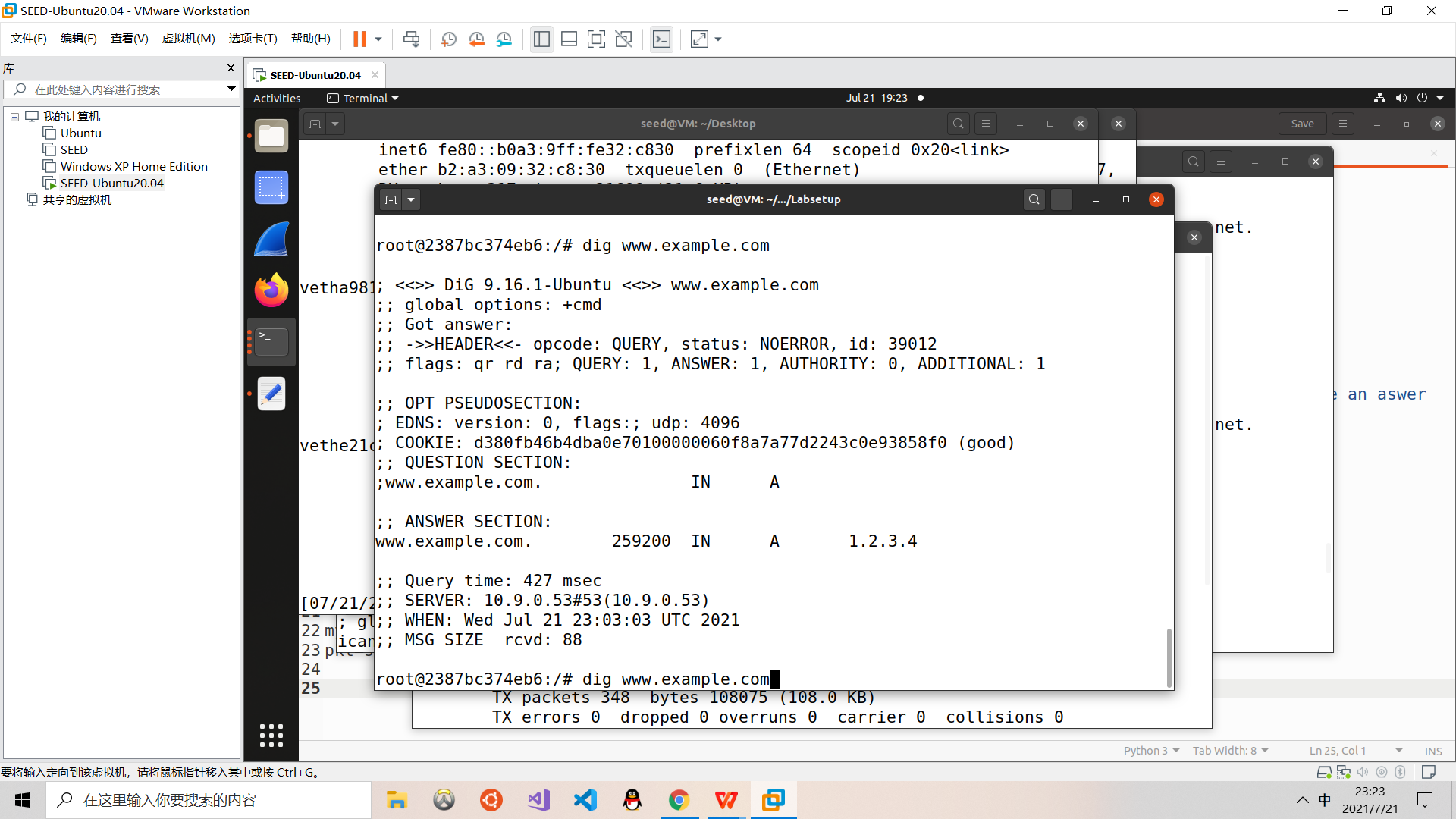


还是之前类似的操作，运行脚本；

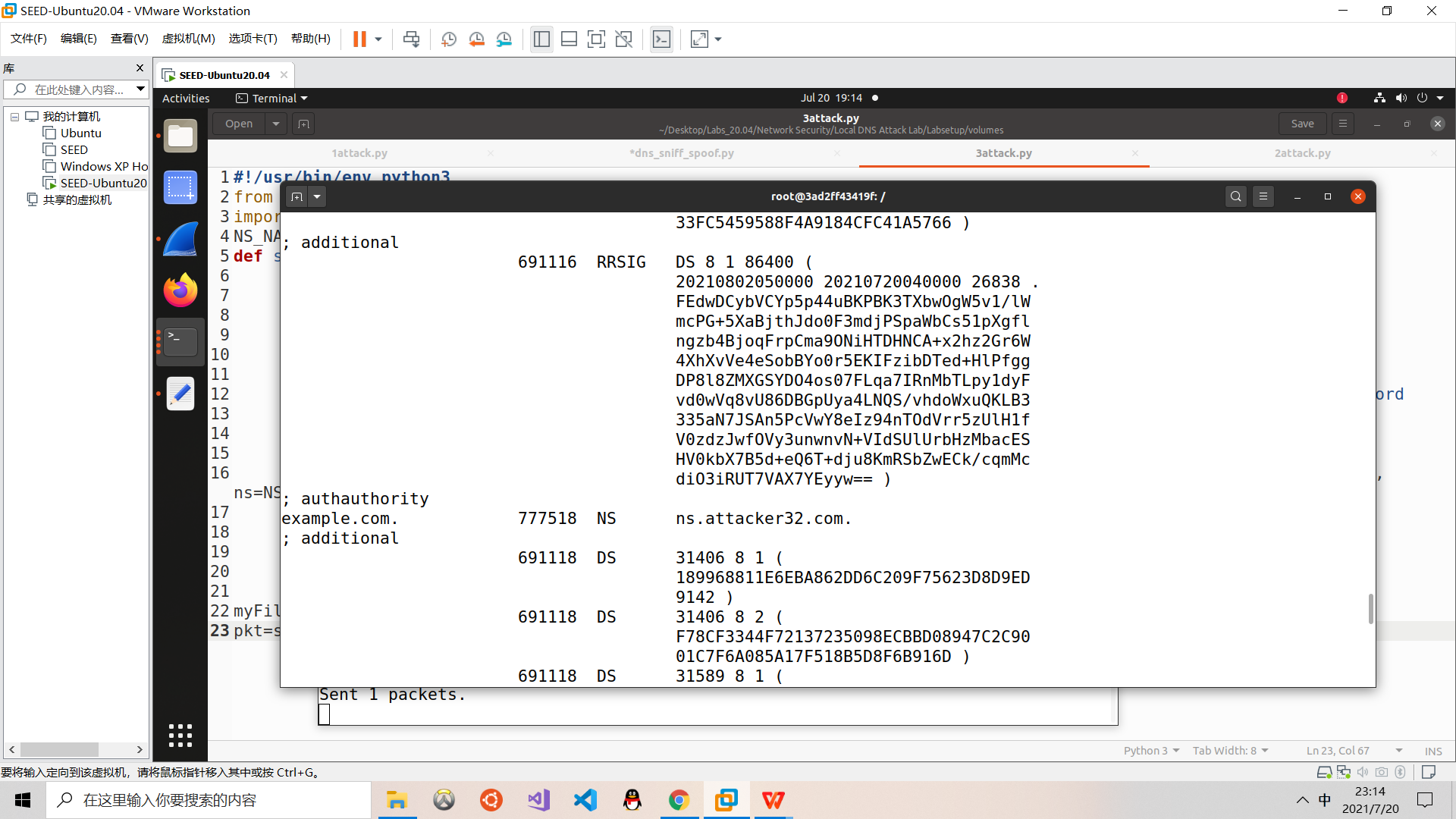
本地DNS服务器上输入 rndc flush，刷新缓存；

在受害者机器上输入命令 dig www.example.com；

在受害这机器上可看见如下内容（如果没有answer但是脚本正常发包，则不刷新缓存再输入命令dig www.example.com）：



本地DNS服务器上先后输入命令rndc dumpdb -cache和cat /var/cache/bind/dump.db，在导出的缓存文件中可以找到如下结果：

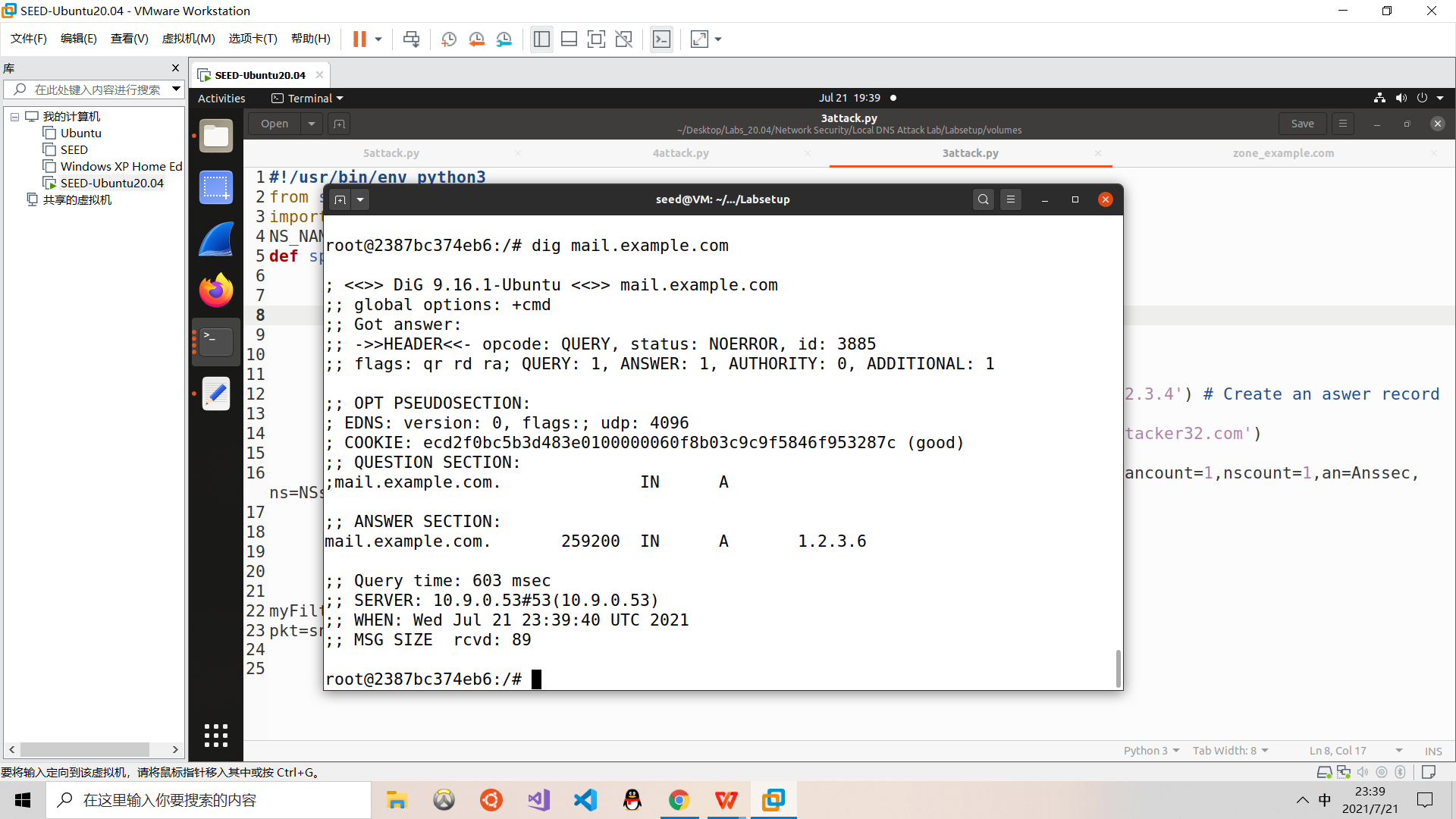


如图所示，攻击成功，已成功对域名www.example.com进行污染。

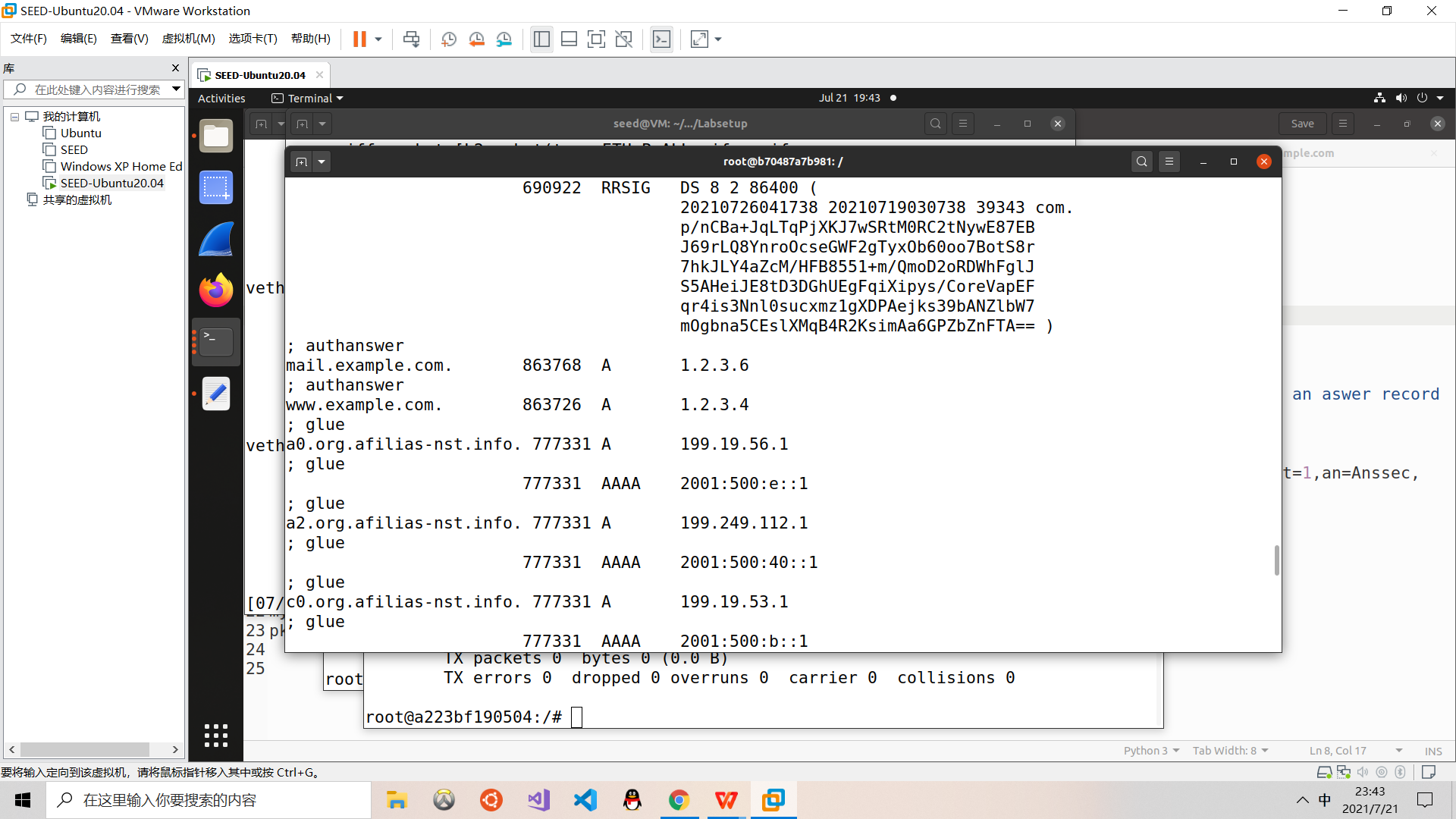
这时候再尝试对域名mail.example.com进行污染；

次在受害者机器上输入命令 dig mail.example.com；

在受害这机器上可看见如下内容：



在本地DNS服务器上先后输入命令rndc dumpdb -cache和cat /var/cache/bind/dump.db，在导出的缓存文件中可以找到如下结果：

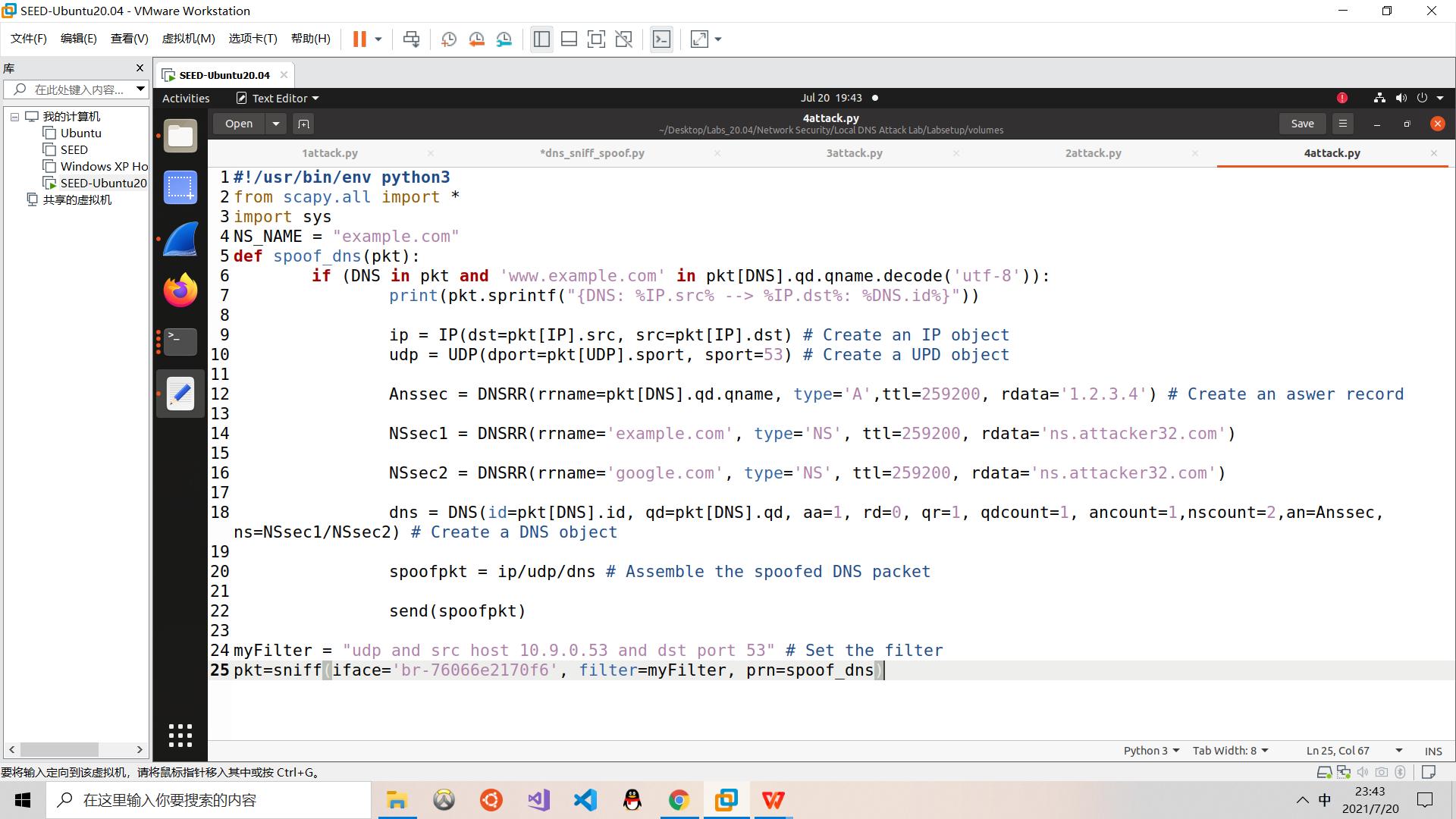


如图所示，攻击成功，已成功对域名www.example.com进行污染。

解析后的域名是1.2.3.6的结果是来自于attacker-router之前的配置。

**Task 4: Spoofing NS Records for Another Domain**

对task3的python程序中负载部分稍做增加，先尝试对本地DNS服务器攻击：

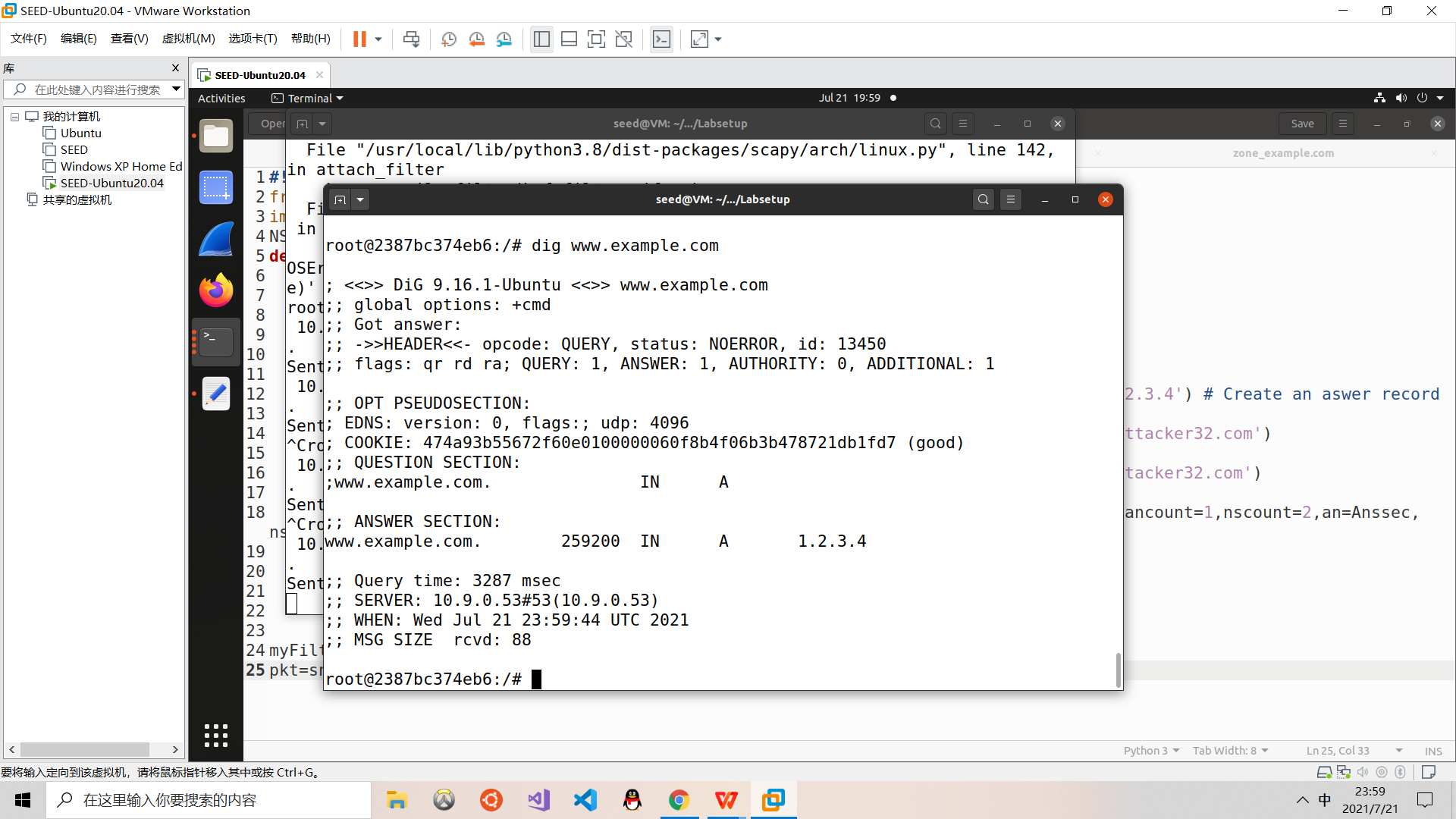


还是之前类似的操作，运行脚本；

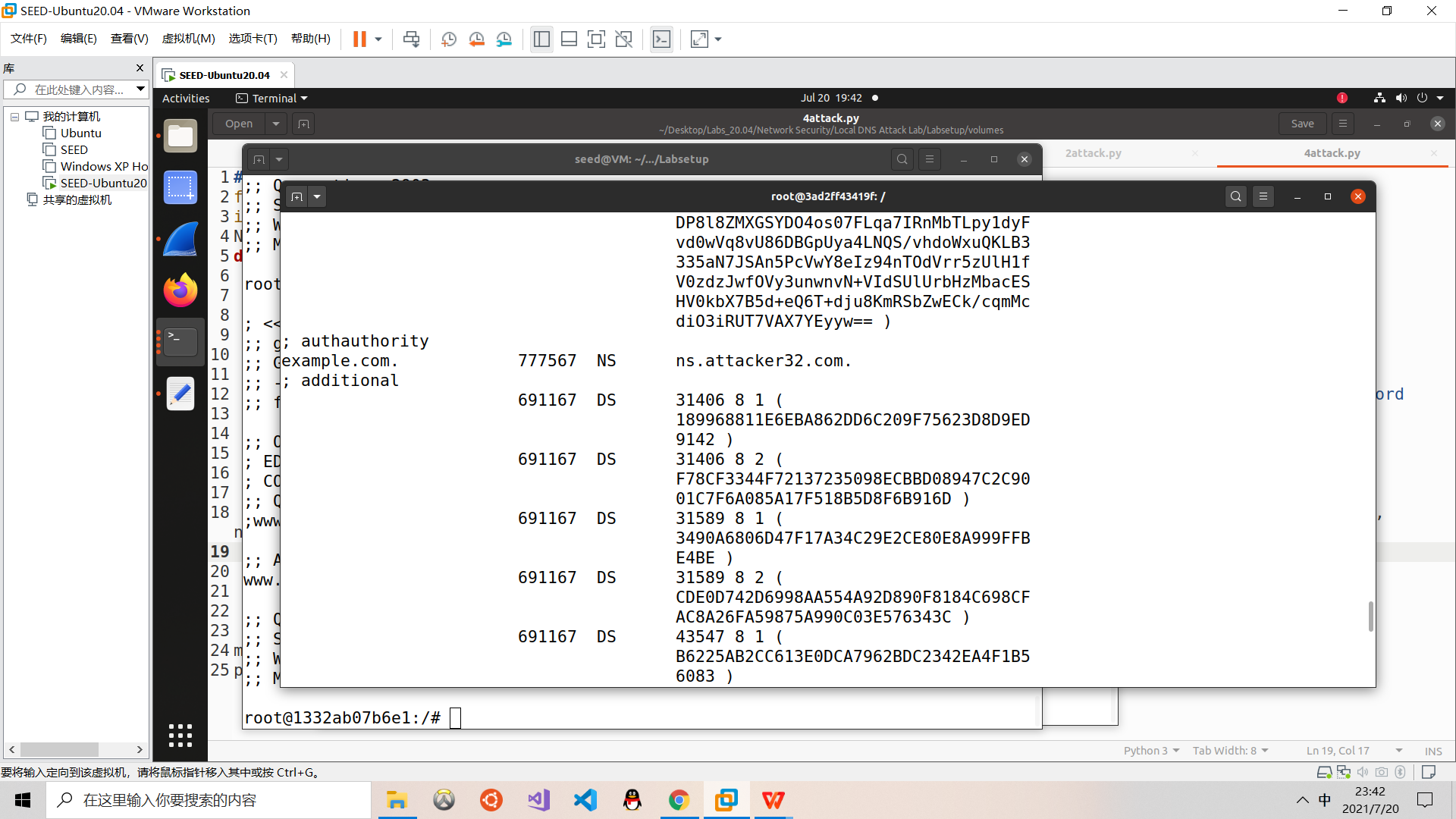
本地DNS服务器上输入 rndc flush，刷新缓存；

在受害者机器上输入命令 dig www.example.com；

在受害这机器上可看见如下内容，根本没有authority section（如果没有answer但是脚本正常发包，则不刷新缓存再输入命令dig www.example.com）：



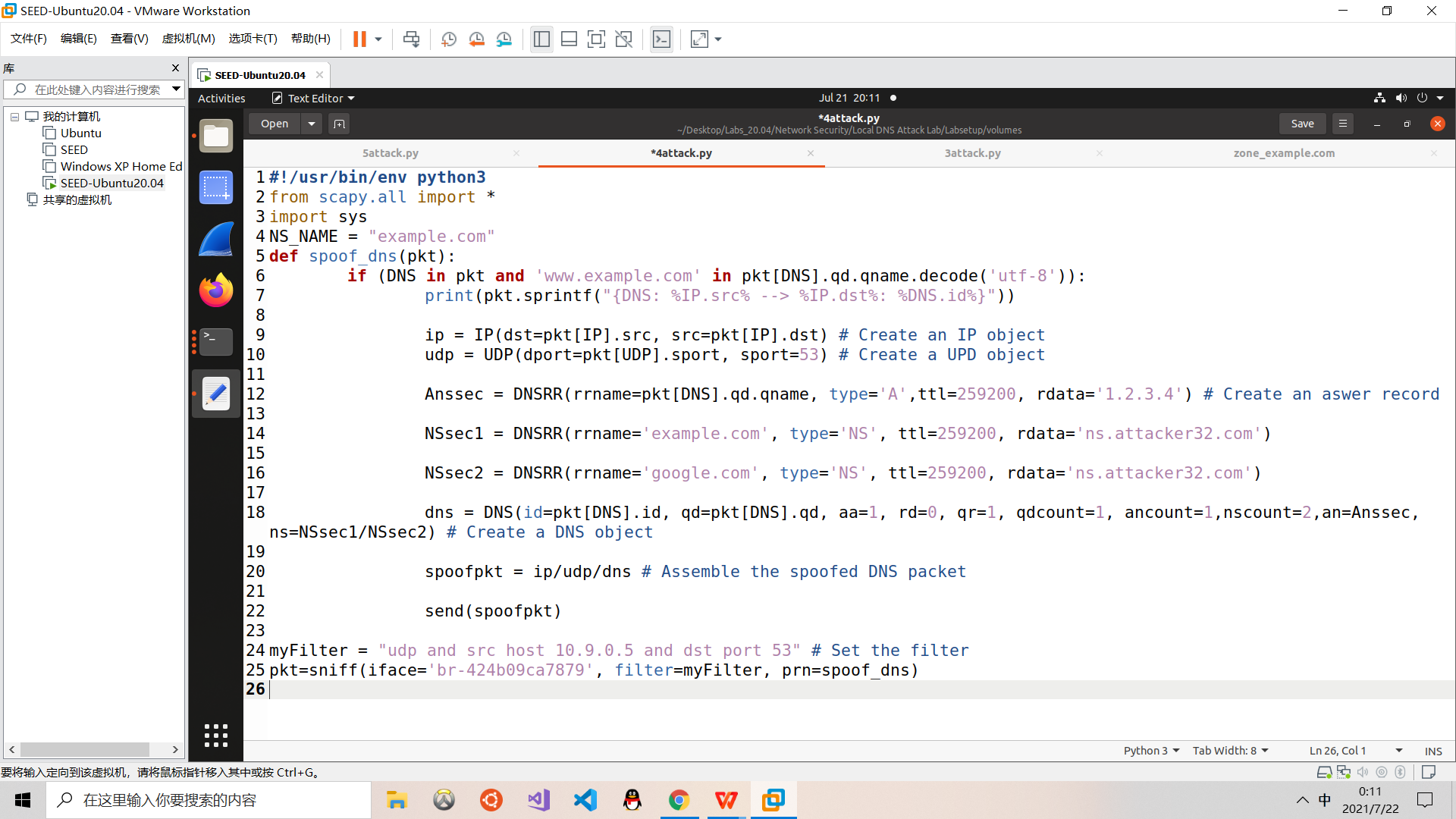
本地DNS服务器上先后输入命令rndc dumpdb -cache和cat /var/cache/bind/dump.db，在导出的缓存文件中可以找到如下结果：



如图，并没有将google.com的域名加入只有example.com的。

这与实验要求得到的结果不吻合。

这时尝试改变filter过滤规则，将过滤规则改为对源地址是10.9.0.5，即对客户机发起攻击而不是对DNS服务器攻击。

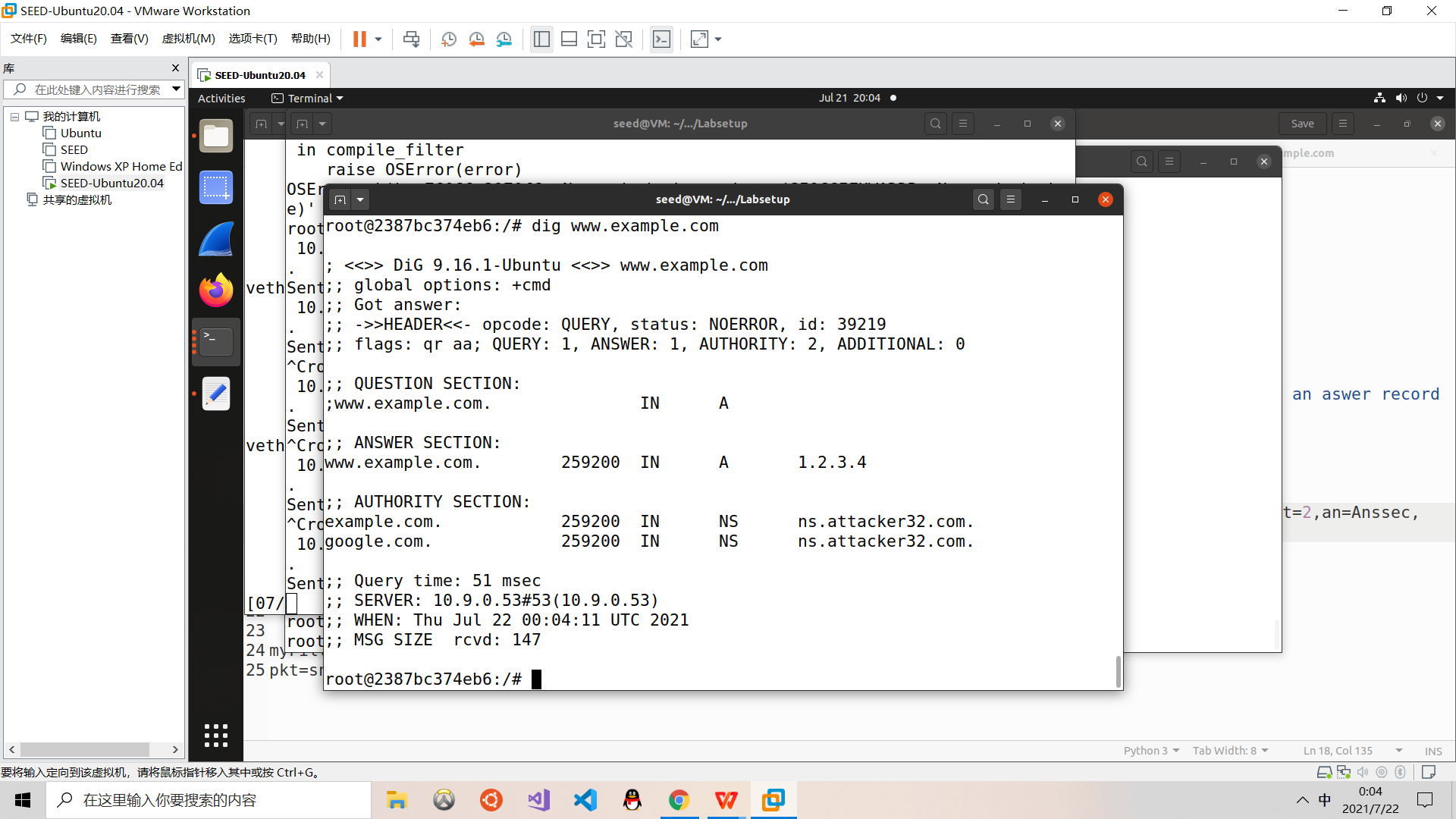


运行修改过滤后的脚本；

本地DNS服务器上输入 rndc flush，刷新缓存；

再次在受害者机器上输入命令 dig www.example.com；

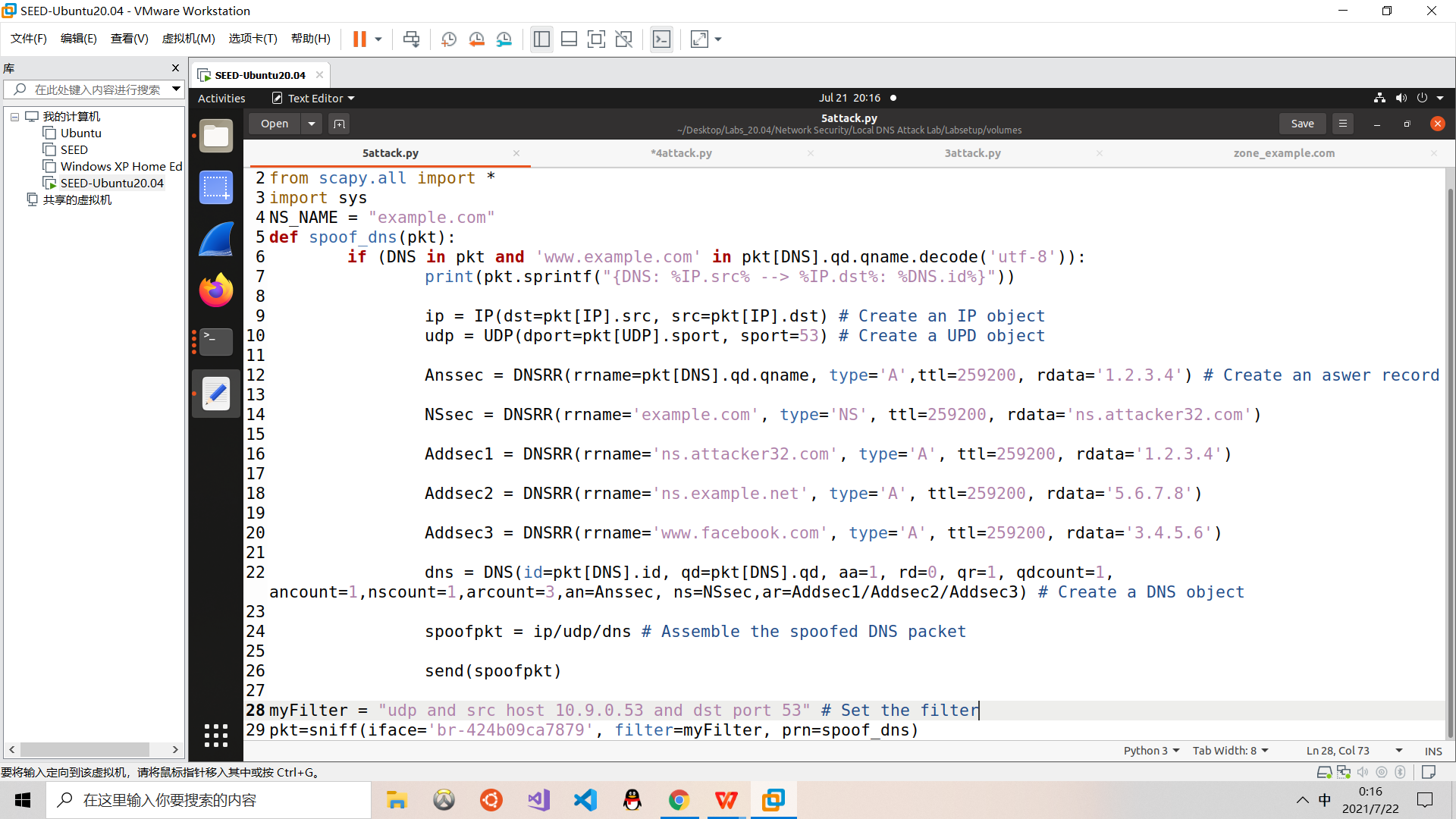
在受害这机器上可看见如下内容，出现了authority section：



之后又进行了额外的尝试，如果将脚本修改，并在10.9.0.5查询google.com，在本地DNS缓存中就会将google.com的域名加入了，但是没有example.com的。

**Task 5: Spoofing Records in the Additional Section**

对task3的python程序中负载部分稍做增加，先尝试对本地DNS服务器攻击：

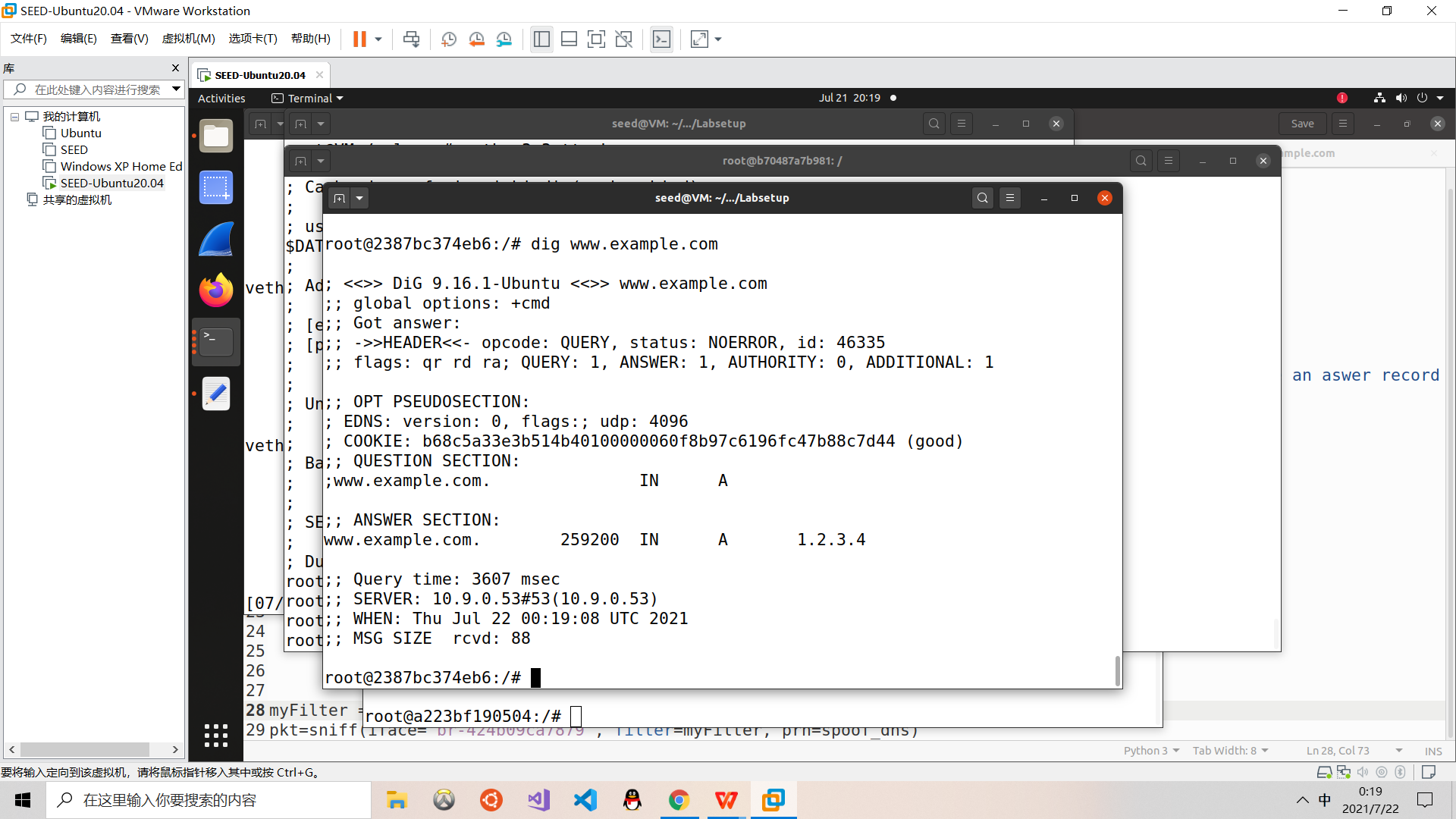


还是之前类似的操作，运行脚本；

本地DNS服务器上输入 rndc flush，刷新缓存；

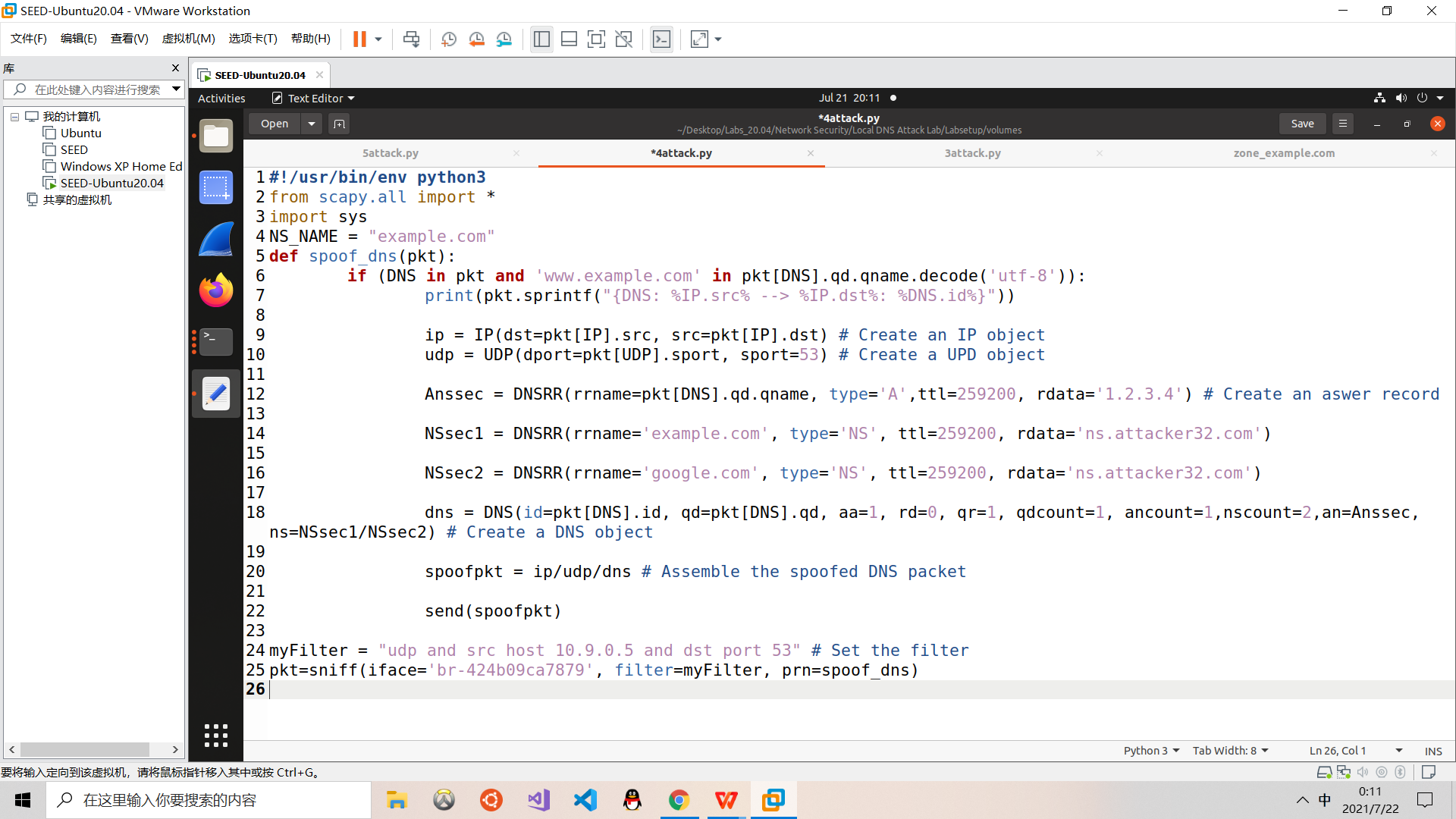
在受害者机器上输入命令 dig www.example.com；

在受害这机器上可看见如下内容，根本没有additional section：



本地DNS服务器上先后输入命令rndc dumpdb -cache和cat /var/cache/bind/dump.db，在导出的缓存文件中也找不到我们写入负载的additional部分。

这与task4情况相似，我们采用和task4同样的手段，改变filter过滤规则，将过滤规则改为对源地址是10.9.0.5，即对客户机发起攻击而不是对DNS服务器攻击。



运行修改过滤后的脚本；

本地DNS服务器上输入 rndc flush，刷新缓存；

再次在受害者机器上输入命令 dig www.example.com；

在受害这机器上可看见如下内容，出现了additional section：

