**Task 6 – DNS Attack**

**环境配置**

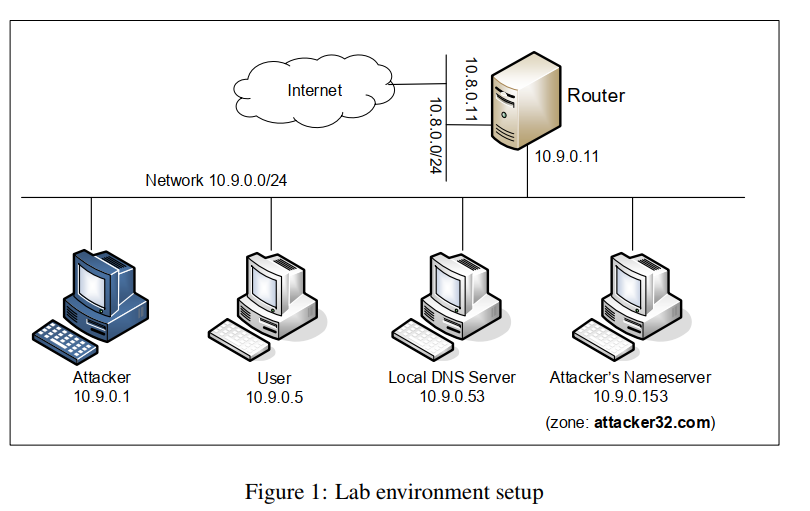
进入实验目录/home/seed/Desktop/Labs\_20.04/Network Security/ Local DNS Attack Lab /Labsetup



另开多个终端，输入doksh xx开启相应容器：分别是router、attacker、user、local dns server、attacker’s nameserver：



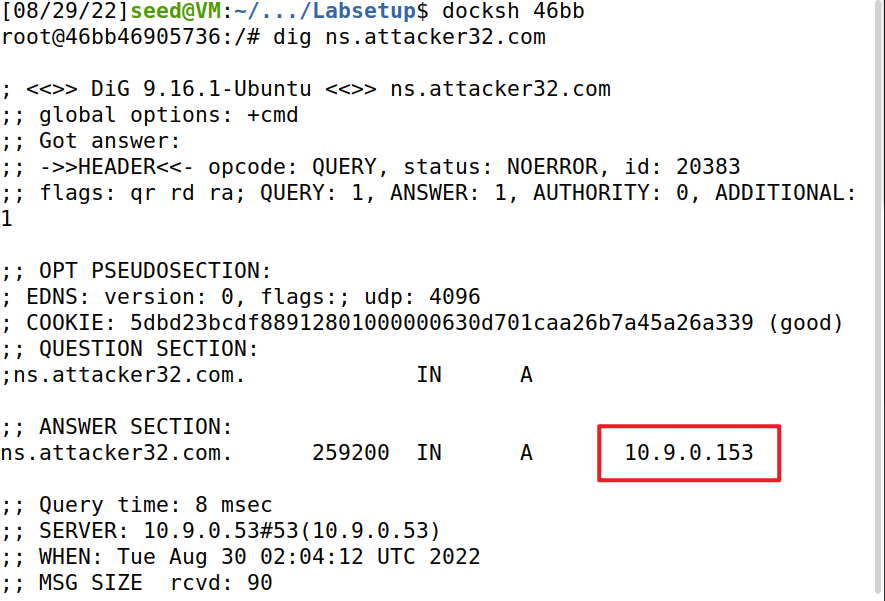
四个主机处于同一个子网中，网络拓扑如下：



1. **Testing the DNS Setup**

在User Container(10.9.0.5)上运行一些命令保证实验环境初始是没有问题的。

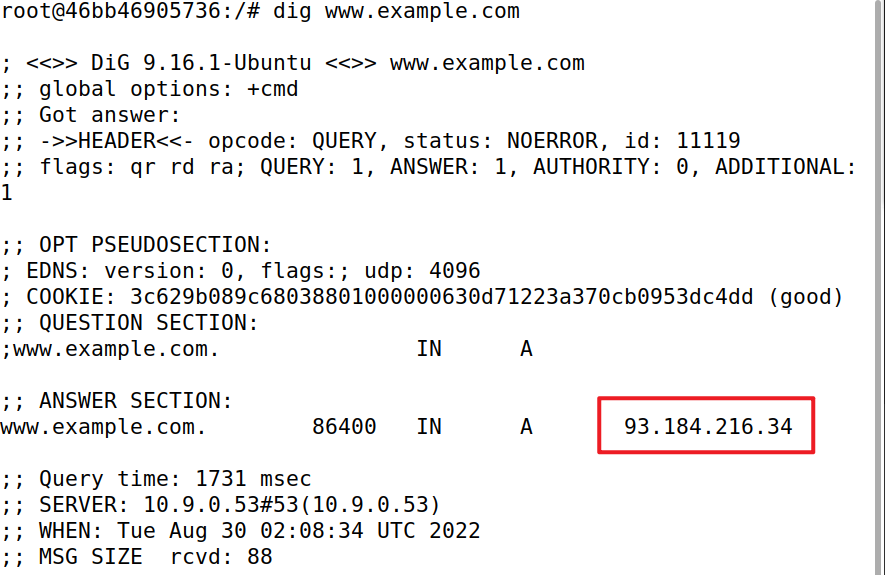
**测试获取ns.attacker32.com的IP地址：**



可以看到结果为10.9.0.153，这一步是通过attacker ns server的，所以应该和attacker ns中/etc/bind/zone\_attacker32.com中的结果是对应的：

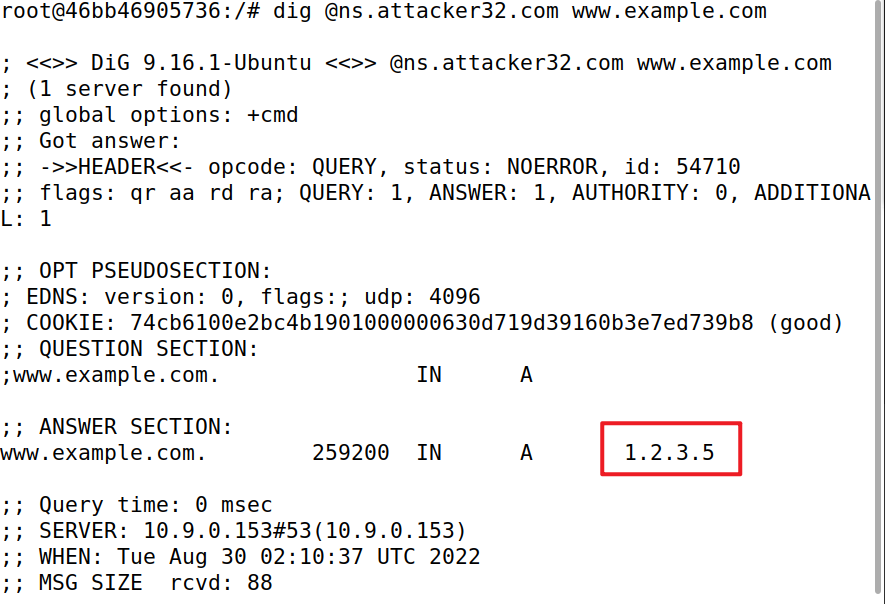


**测试获取www.example.com的IP地址：**



这一步没有经过attacker ns server，所以结果应该是正确的结果。

**测试经过attacker nameserver查询www.example.com的IP地址：**

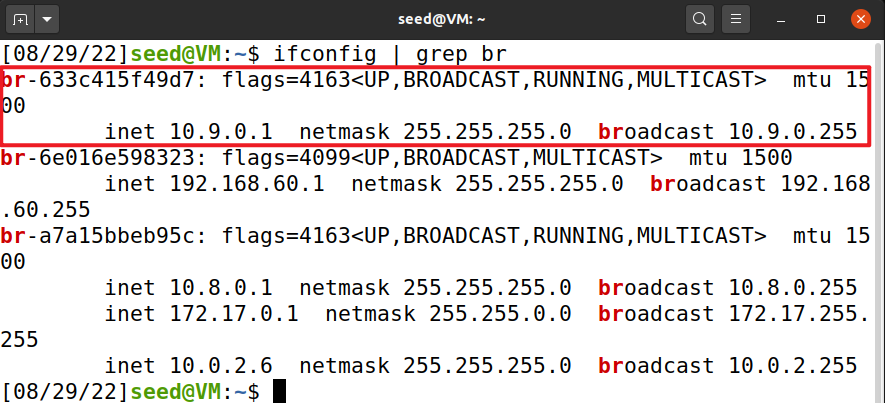


地址变为了1.2.3.5，这一步经过了attacker ns server查询www.example.com，所以查询到的结果应该是attacker nameserver的/etc/bind/zone\_example.com中的结果：



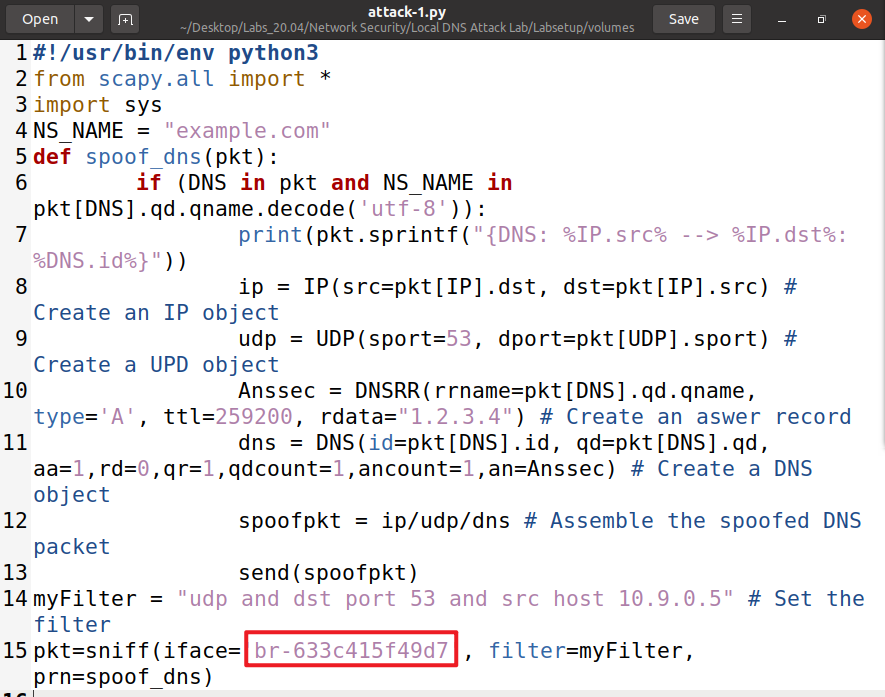
1. **Directly Spoofing Response to User**

虚拟机中输入ifconfig | grep br，查看10.9.0.0/24网段interface名称：



对应的是br-633c415f49d7。

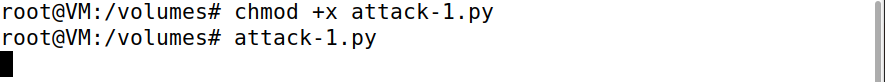
在attacker主机上创建attack-1.py文件，内容如下：



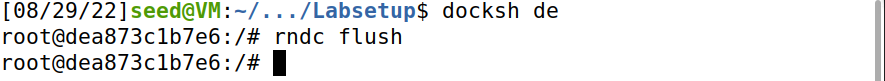
注意要修改为自己机器的网卡名称。

这里收到10.9.0.5的DNS请求的包后，[如果发现是查询example.com](http://xn--example-hk3l463bg14axjddzak53ix10e.com)，将回复1.2.3.4。

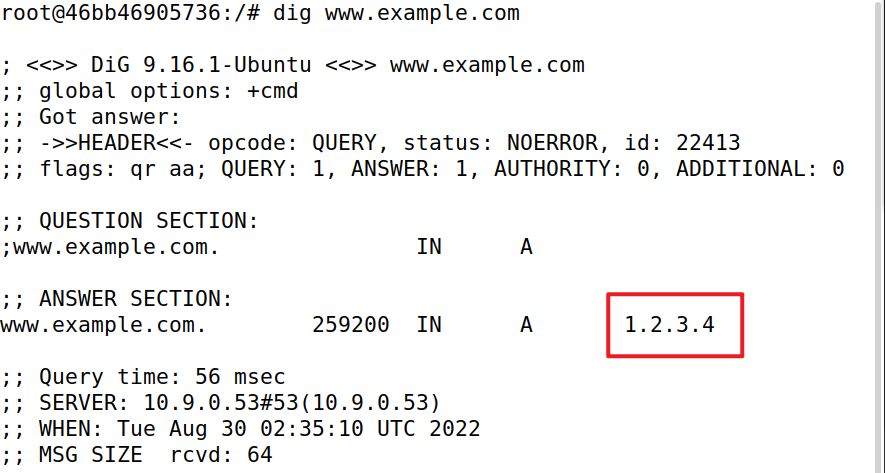
输入chmod +x attack-1.py添加执行权限, 然后执行attack-1.py开启监听:



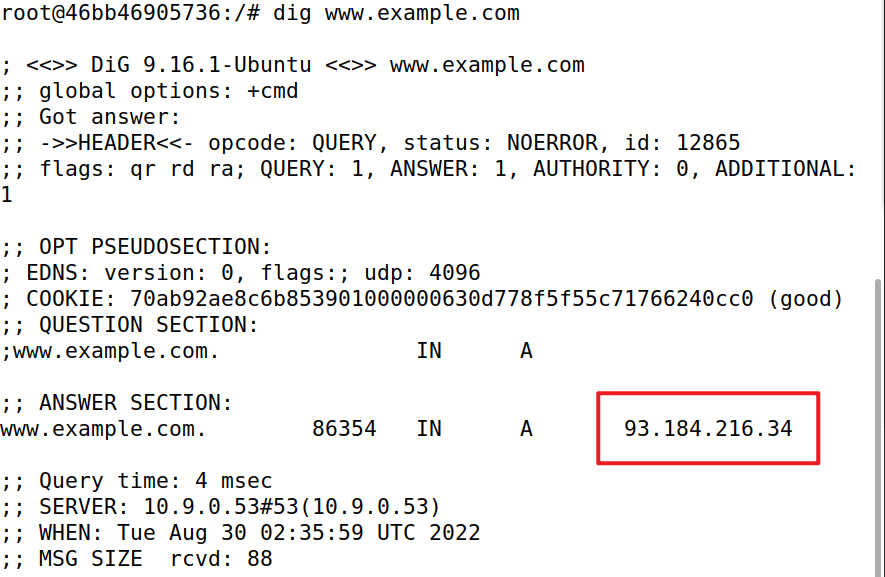
在dns server上(10.9.0.53)输入rndc flush清除cache，否则其返回是比attacker要快的，我们就无法成功攻击了：



在User容器上执行对www.example.com的查询，结果如下:



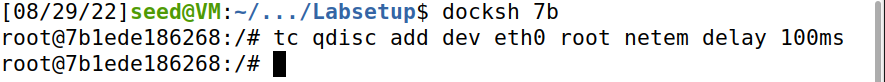
可以看到变成了1.2.3.4，再次进行查询时，因为dns server已经查询过，cache中有了记录，所以攻击失败，获取到的是正确的地址：



注意: 这里因为网络访问的DNS请求甚至都比容器间的块，所以我们直接在Router容器中添加到外网的访问延时，这样在后续实验中就可以不受此问题困扰了。

解决方法如下:

在Router容器中使用以下命令为eth0外网网卡添加延时:



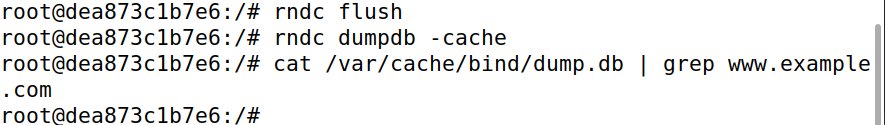
1. **DNS Cache Poisoning Attack – Spoofing Answers**

攻击User主机效率太低了，每次都要响应User主机的请求，不如直接攻击DNS server，可以一劳永逸，缓存会在local dns server上保存很久。

在local dns server上使用rnds flush命令清空DNS cache，然后使用以下命令查看DNS缓存：

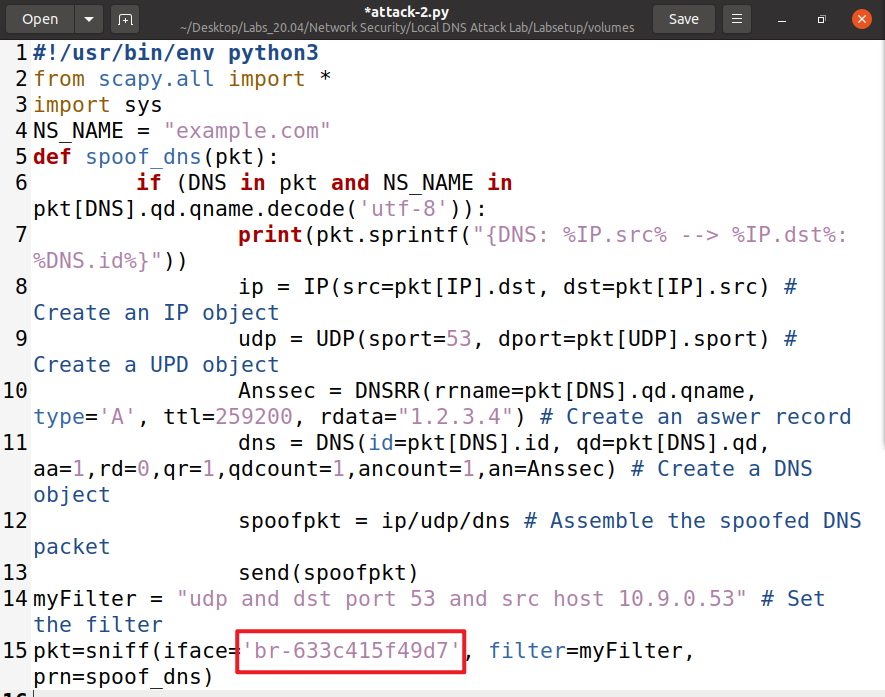
rndc dumpdb -cache：将缓存导出到特定文件

cat /var/cache/bind/dump.db：查看文件



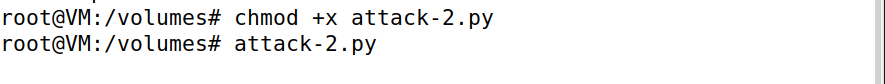
可以看到当前local dns server没有www.example.com缓存。

在Attacker主机上创建attack-2.py文件，内容如下:

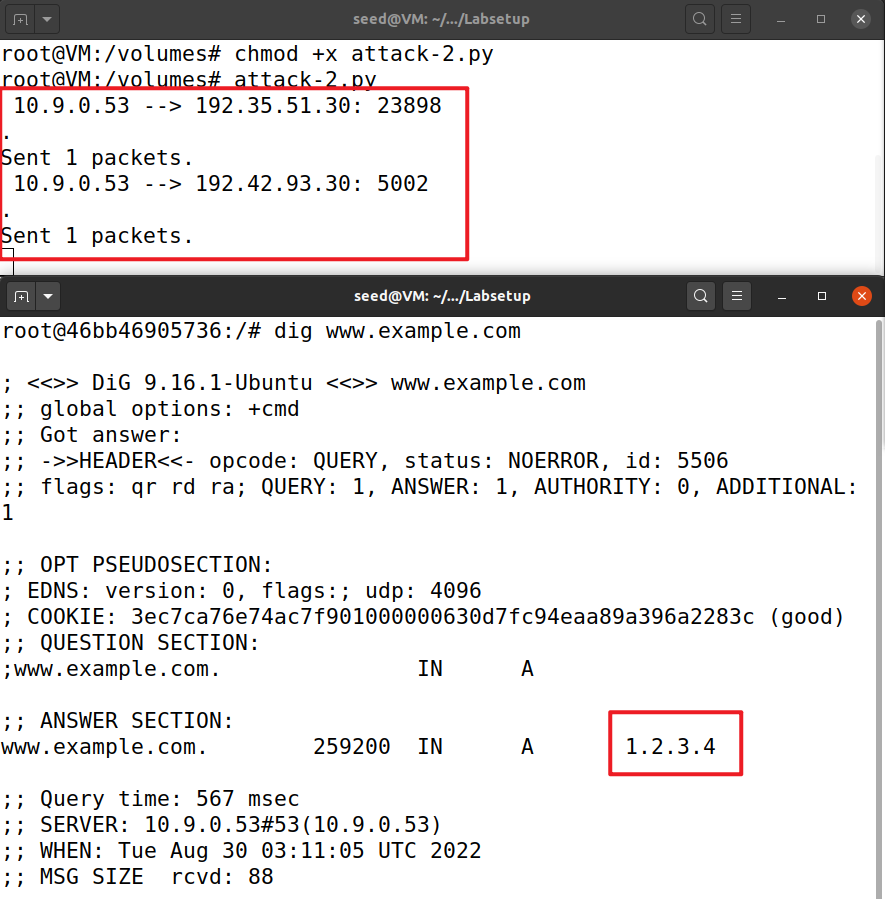


还是注意修改网卡名称。

添加运行权限后运行脚本：

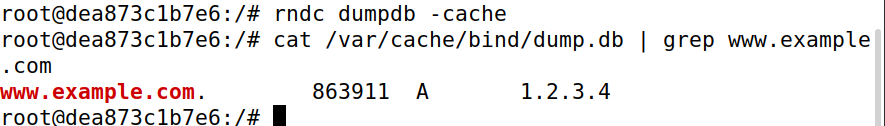


在User主机上执行查询命令，结果如下:



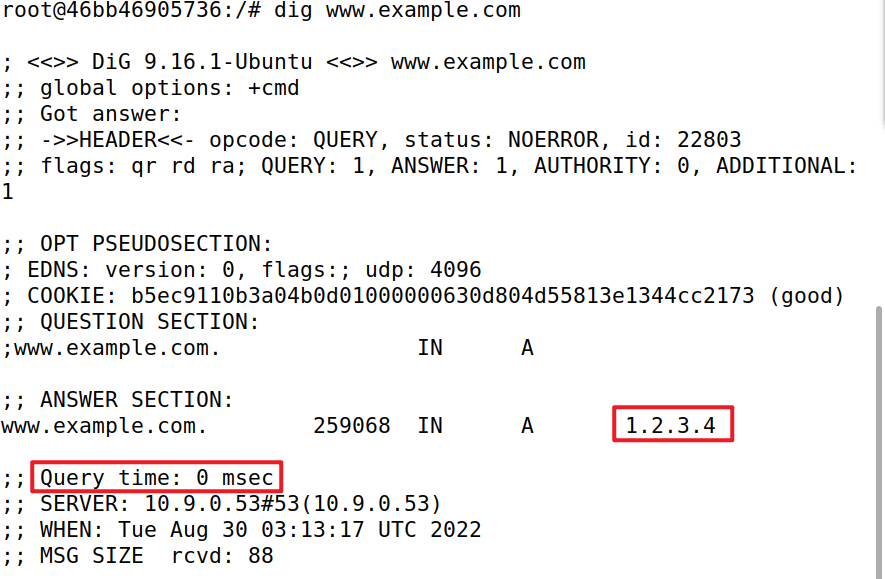
可以看到attacker也发出了欺骗的包。

在local dns server上查看DNS 缓存，结果如下:



可以看到确实拥有了www.example.com的缓存。

之后在关掉Attacker主机上的attack-2.py程序之后，在User主机上查询就直接得到1.2.3.4的结果了：

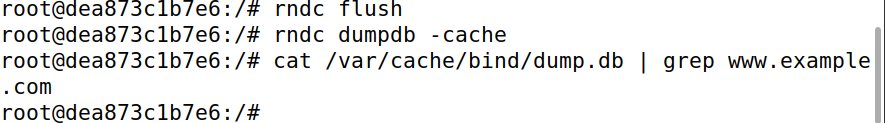


并且可以看到请求时间为0ms，通过时间判断确实是直接取的local dns server上的缓存。

1. **Spoofing NS Records**

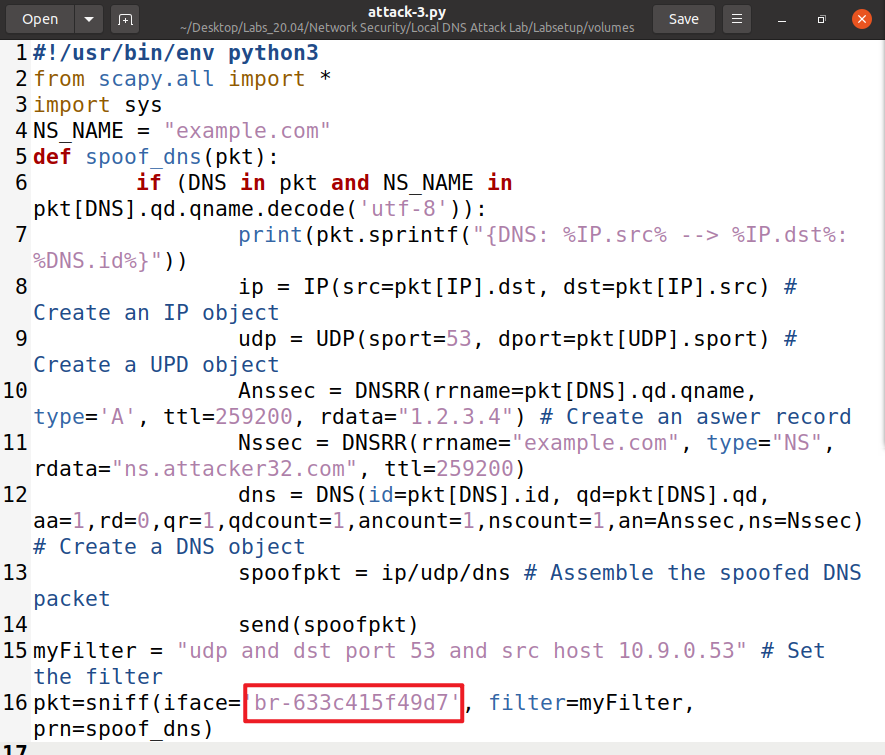
上面的实验只能修改www.example.com的结果，这个Task要一次可以修改example.com整个域的结果。

先清空local dns server的DNS cache。



可以看到已经为空。

在Attacker上创建attack-3.py，内容如下：

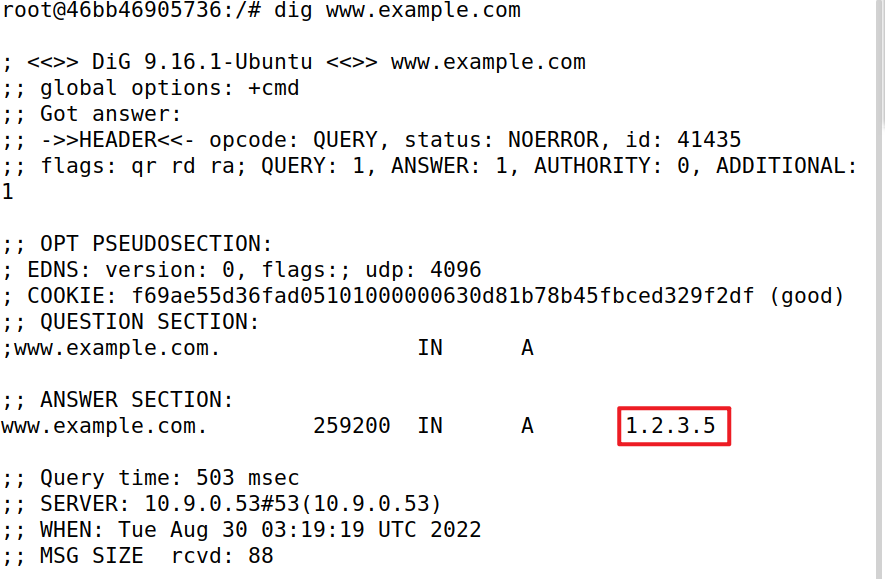


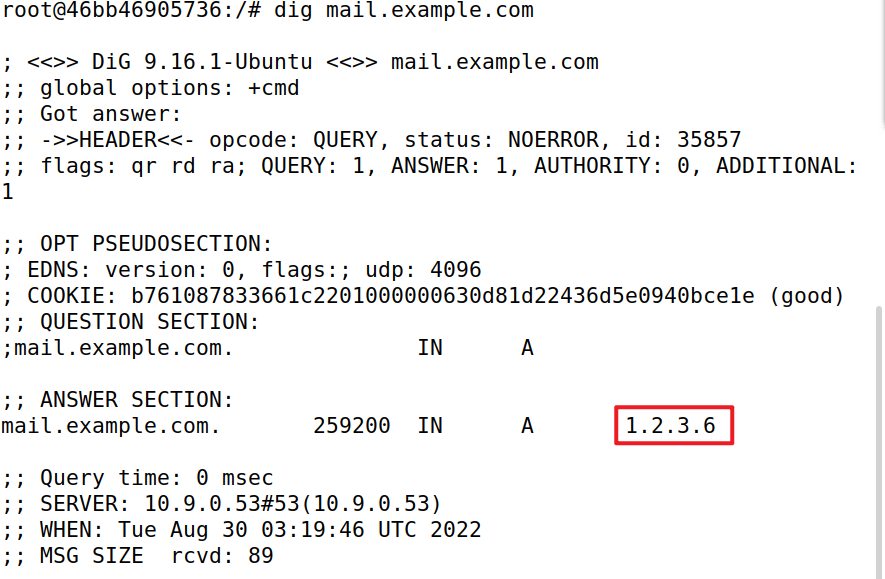
注意修改网卡名称。

添加执行权限后运行：

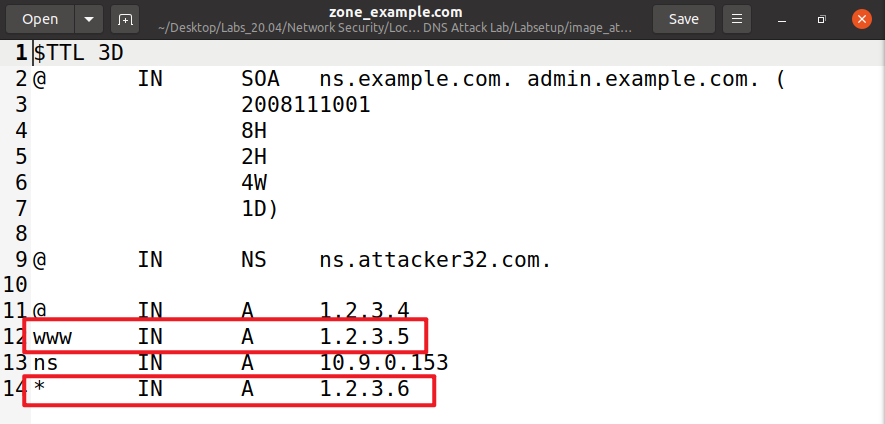


在User主机上查询www.example.com和mail.example.com的结果:



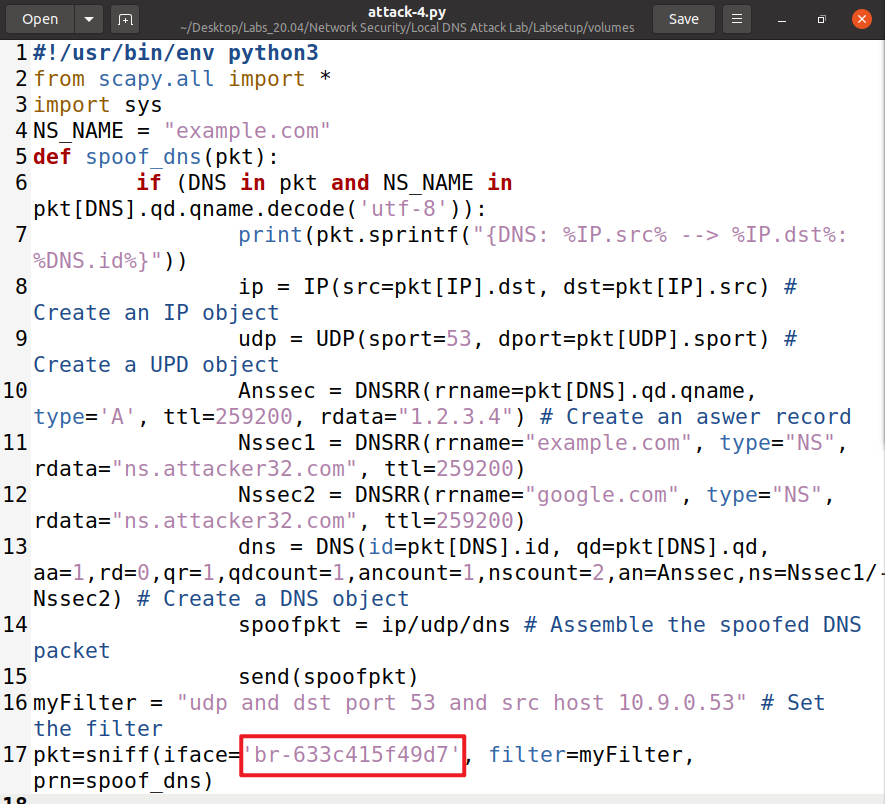


分别是1.2.3.5和1.2.3.6，和attacker dns server上的结果一致：

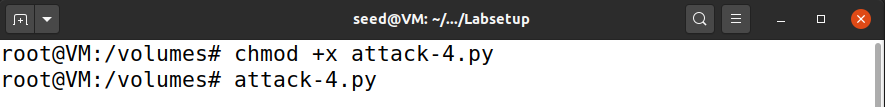


1. **Spoofing NS Records for Another Domain**

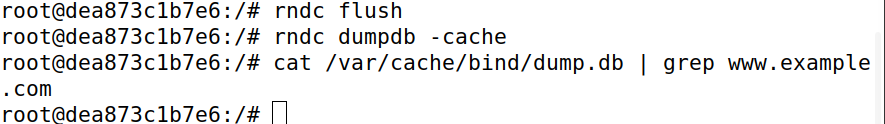
在Attacker上创建文件attack-4.py，内容如下：



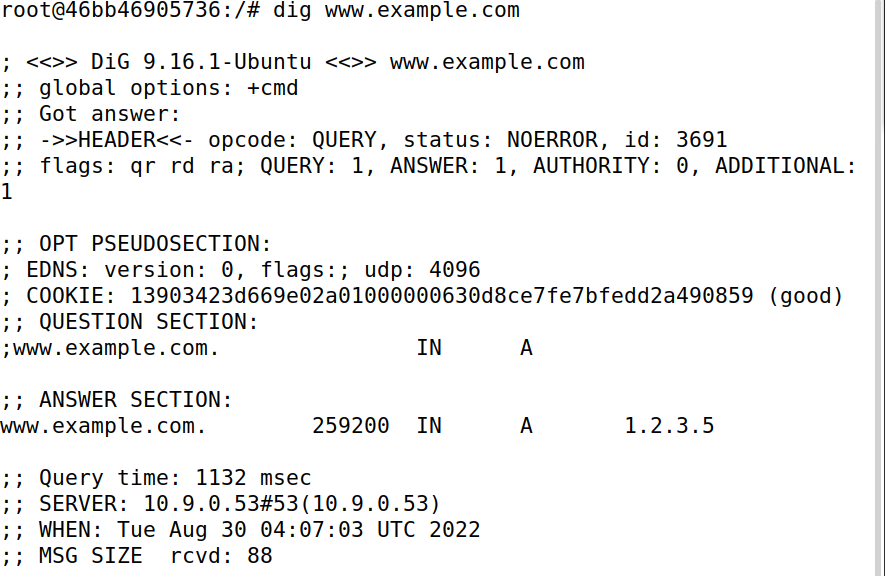
添加执行权限后运行脚本：



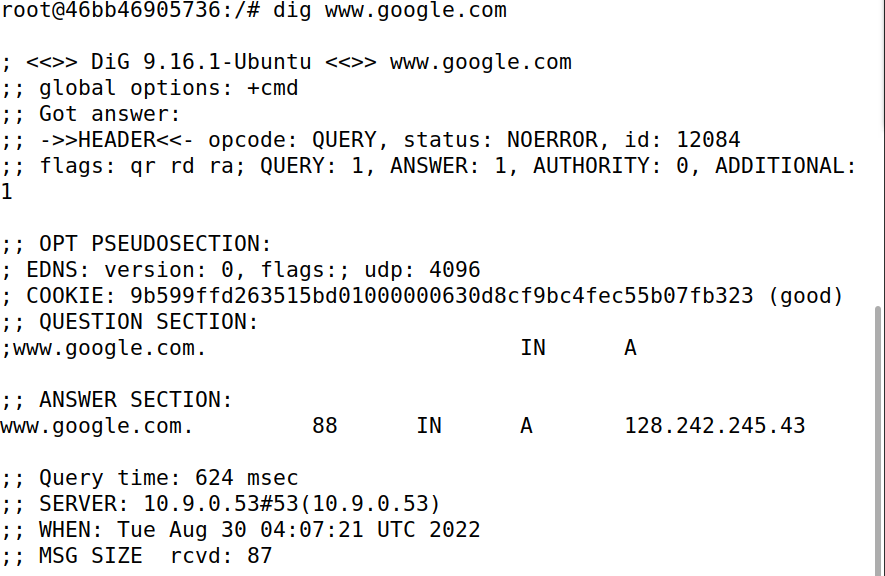
在local dns server中清除缓存：



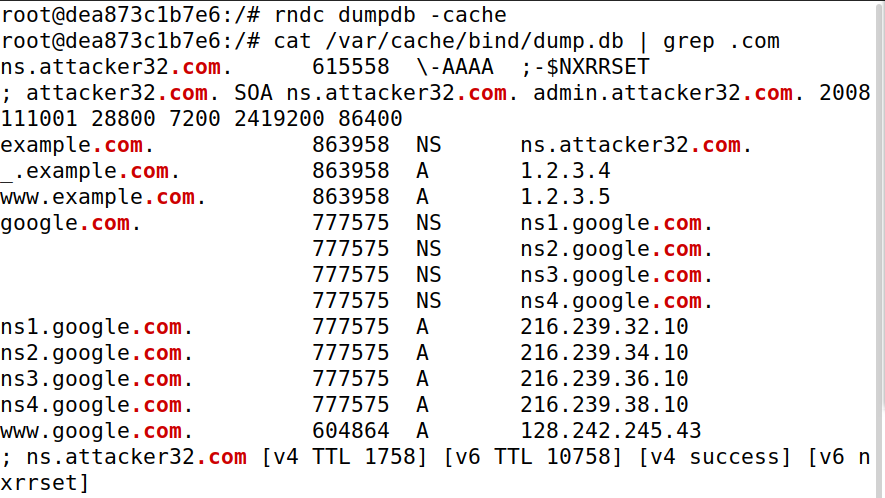
在User主机中查询www.example.com来完成攻击(因为查询example.com才会受到攻击)，结果没问题。



之后查询www.google.com，发现并未使用ns.attacker32.com作为其DNS



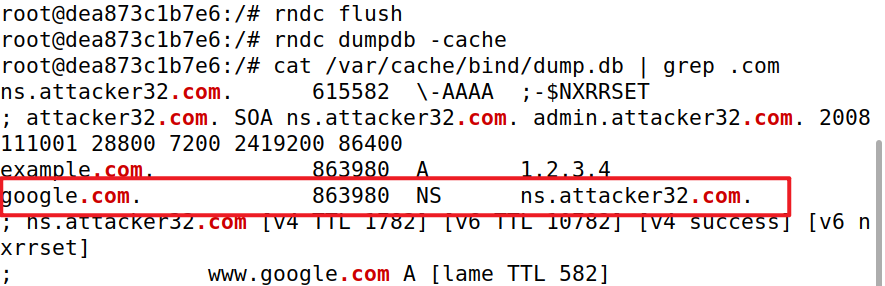
查询local dns server DNS缓存之后结果如下：



查询时并没有经过ns.attacker32.com。

分析原因是权威字段中只能有一条有效，且是排在前面的一条有效，下面来证实以上分析。

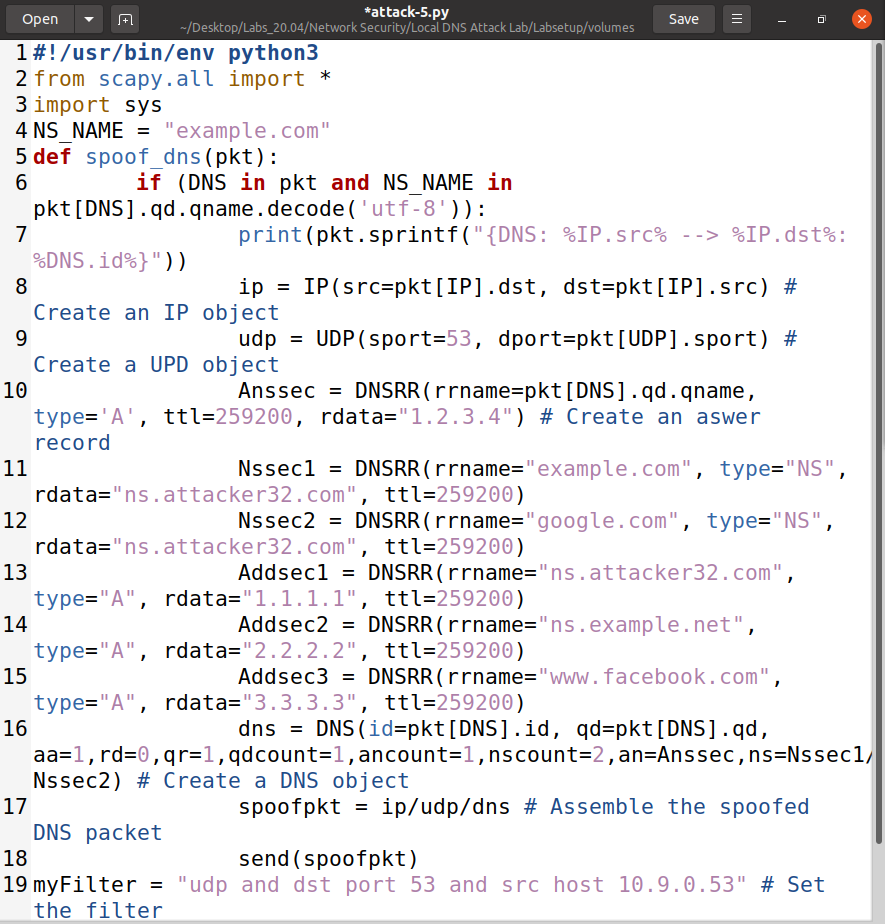
将attacke-4.py文件中Nssec两行顺序对调后（即Nssec1对应google.com，Nssec2对应example.com），清空local dns server缓存内容，再次攻击，可以看到这次google.com查询的服务器变为了ns.attacker32.com。如下：



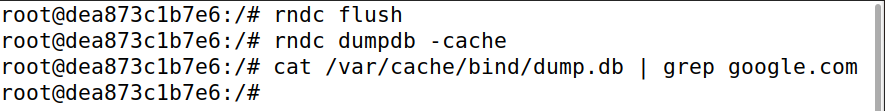
所以由此看出在一次攻击中伪造多条记录时不可行的。

1. **Spoofing Records in the Additional Section**

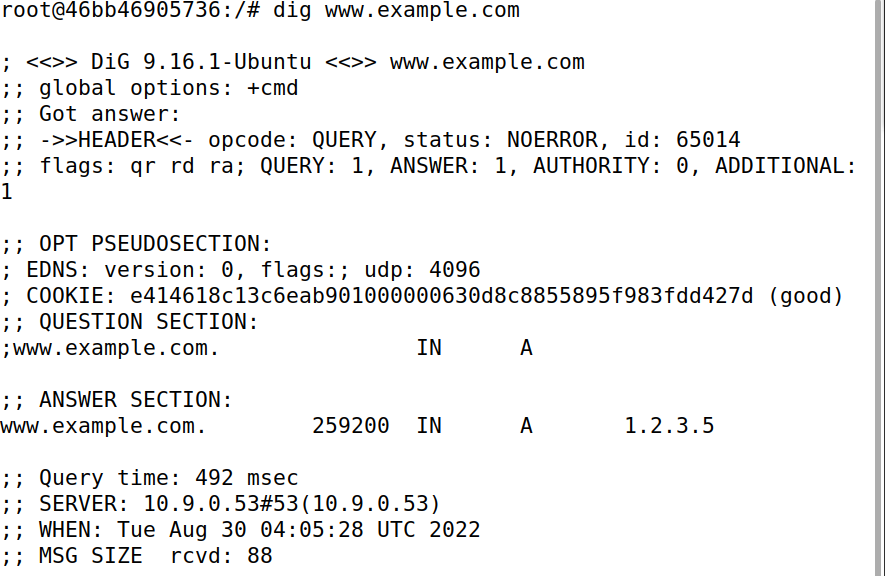
在Attacker上创建文件attack-5.py，内容如下：



在local dns server中清空缓存：

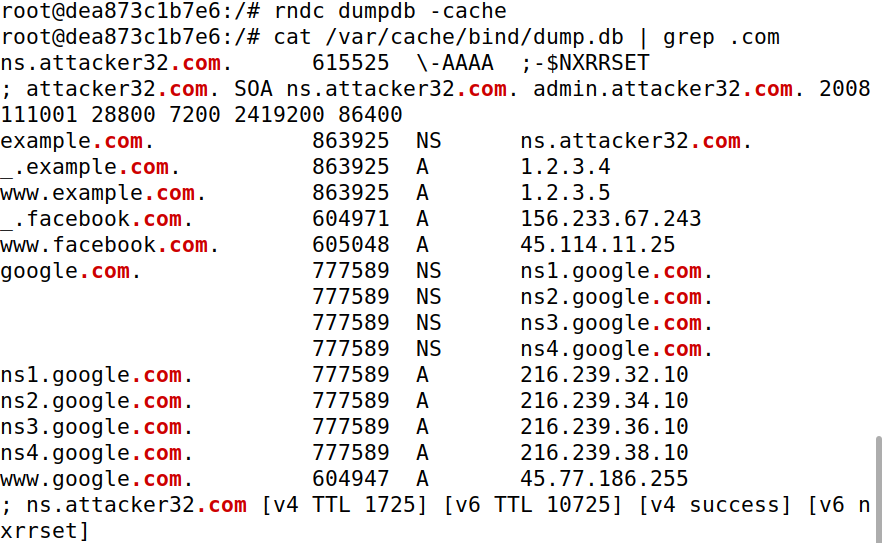


在User主机中查询www.example.com来完成攻击(因为查询example.com才会受到攻击)，结果没问题：



之后并使用dig命令来查询攻击脚本中所写的其他域名的ip地址，包括ns.facebook.com、ns.example.net等，发现均不是我们所写入的地址。

查看locsal dns server的DNS cache，结果如下:



只有example.com是通过ns.attacker32.com来查询的，说明仍然是只有权威字段中的内容有效，而附加字段中的内容并不会存入缓存中。