Name: Kazi Noshin

Student ID: 1605071

Attack Tool 19: DNS cache poisoning + Phishing attack

Contents:

Steps of Attack:

Environment setup:

To demonstrate this attack, I needed three machines.

- A DNS server
- Victim user
- Attacker

Here I used docker from seedlabs (since there is not enough space in my laptop to run three virtual machines at a time).

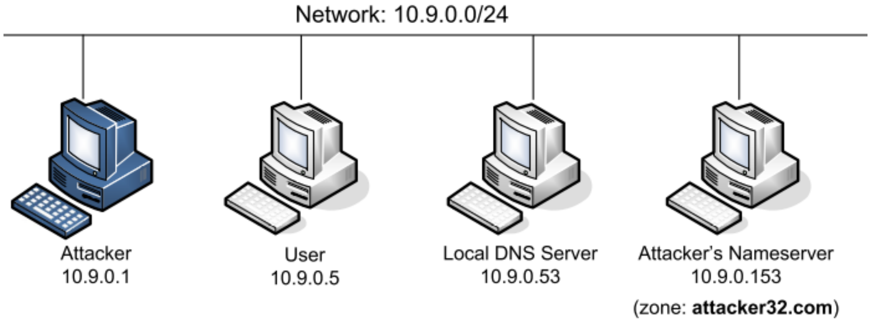


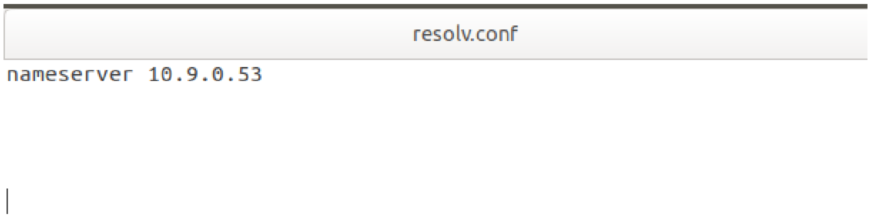
Figure: Environment setup

I used the following two commands to open the containers:

- *sudo docker ps*
- *sudo docker exec -it <id> /bin/bash*

User Configuration:

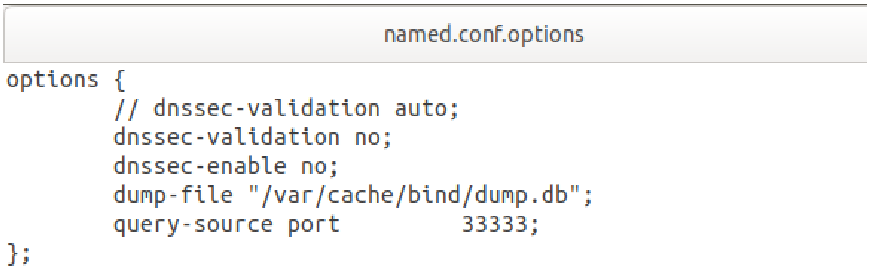
- On the user machine (**10.9.0.5**), **10.9.0.53 is used** as the local DNS server.
- This is achieved by changing the DNS setting file (**resolv.conf**) of the user machine.



```
nameserver 10.9.0.53
```

Local DNS Server Configuration:

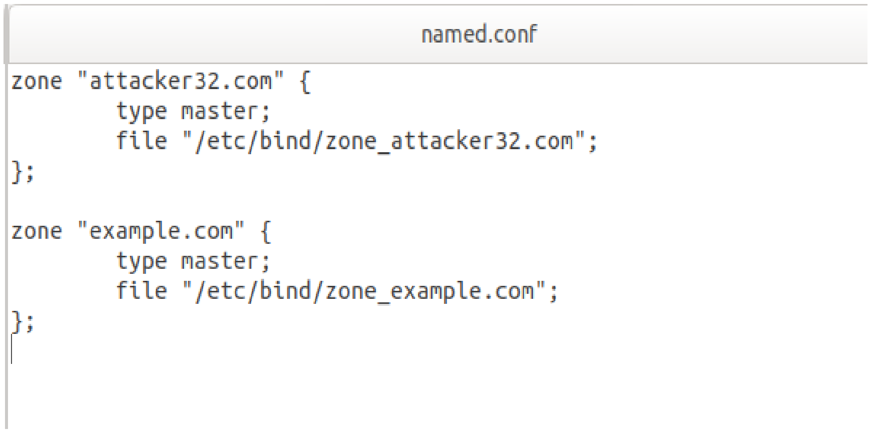
- Bind9 server was used to set up the local DNS.
- To configure the Bind9 server, I edited the file **named.conf.options**.



```
options {
        // dnssec-validation auto;
        dnssec-validation no;
        dnssec-enable no;
        dump-file "/var/cache/bind/dump.db";
        query-source port         33333;
};
```

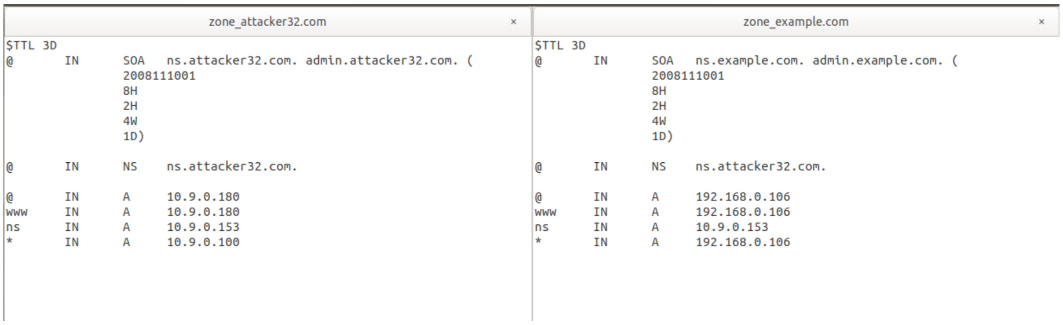
Attacker Machine Configuration:

- I used www.example.com(actual IP: 93.184.216.34) as the attacking target domain.
- I have a fake server named ns.attacker32.com(**10.9.0.153**) on the attacker's machine.
- I needed to create two zone entries in the DNS server by editing the **named.conf** file.



```
zone "attacker32.com" {
        type master;
        file "/etc/bind/zone_attacker32.com";
};

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

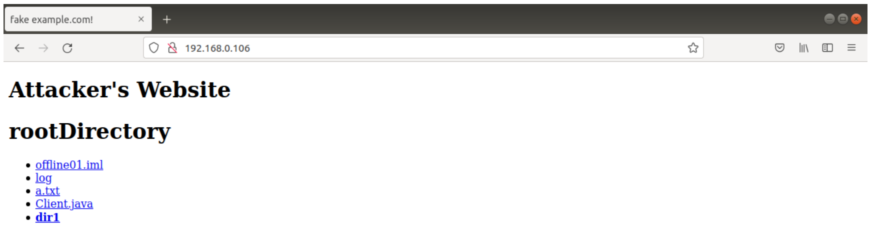
- There are **zone_attacker32.com** and **zone_example.com** files.



```
$TTL 3D
@       IN      SOA   ns.attacker32.com. admin.attacker32.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.attacker32.com.

@       IN      A     10.9.0.180
www     IN      A     10.9.0.180
ns      IN      A     10.9.0.153
*       IN      A     10.9.0.100
```

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

@       IN      NS    ns.attacker32.com.

@       IN      A     192.168.0.106
www     IN      A     192.168.0.106
ns      IN      A     10.9.0.153
*       IN      A     192.168.0.106
```

- I also have a website that is hosted in the IP provided by the attacker.

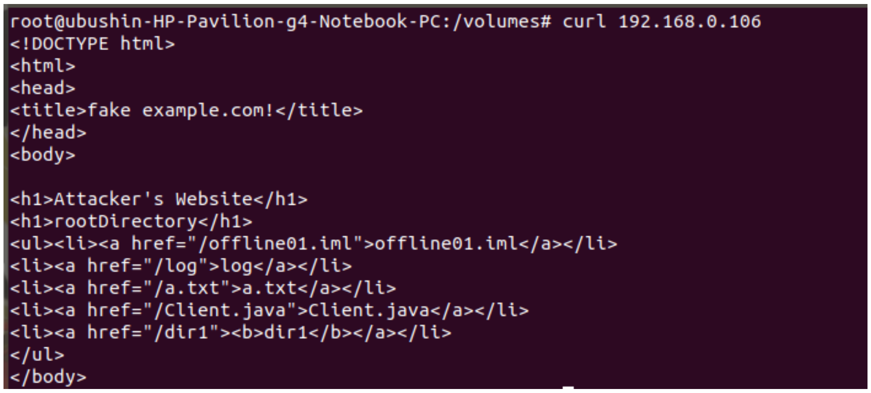


Attacker's Website

rootDirectory

- offline01.iml
- log
- a.txt
- Client.java
- **dir1**

(Since I used docker, I used the ‘**curl**’ command to receive data from a server.)



```
root@ubushin-HP-Pavilion-g4-Notebook-PC:/volumes# curl 192.168.0.106
<!DOCTYPE html>
<html>
<head>
<title>fake example.com!</title>
</head>
<body>

<h1>Attacker's Website</h1>
<h1>rootDirectory</h1>
<ul><li><a href="/offline01.iml">offline01.iml</a></li>
<li><a href="/log">log</a></li>
<li><a href="/a.txt">a.txt</a></li>
<li><a href="/Client.java">Client.java</a></li>
<li><a href="/dir1"><b>dir1</b></a></li>
</ul>
</body>
```

Attack Goals:

- When the user runs the dig command to find out the www.example.com’s IP address, the local DNS server will go to the attacker’s name server ns.attacker32.com to get the IP.
- Then the attacker will be able to give the DNS an IP address as he wishes.
- As a result, the DNS will give the malicious IP to the user, and thus the user will go to the attacker’s website.

Attack Synopsis:

I launched the **Kaminsky attack** to demonstrate DNS cache poisoning.

- The attacker sends a query for a non-existing name in example.com (dferj.example.com) to the DNS server.
- As there is no mapping in the DNS cache, it queries the nameserver for the fake domain.
- While the DNS server waits for the reply, the attacker sends a flood of spoofed DNS responses with a fake IP, an assumed query ID, and setting 1 to the ‘AA’ in the header.

Attack Implementation:

- A random subdomain(which may not exist) is generated.
- A flood of requests is sent to the local DNS server using **send_dns_request** function.
    - DNS query packet is created using the following functions:

Function Name	Function Activity
get_DNS_header	● Fills the DNS header fields with appropriate values
set_ques_record	● Sets a question record to the DNS packet

    - DNS query packet is sent using **send_packet** function.
- A flood of spoofed responses is sent to the local DNS server using **send_DNS_response** function.
    - DNS response packet is created using the following functions:

Function Name	Function Activity
get_DNS_header	(Same as previously mentioned)
set_ques_record	(Same as previously mentioned)
set_resrc_record	● Sets an appropriate resource record to the DNS packet

    - DNS response packet is sent using **send_spoofed_packet** function.

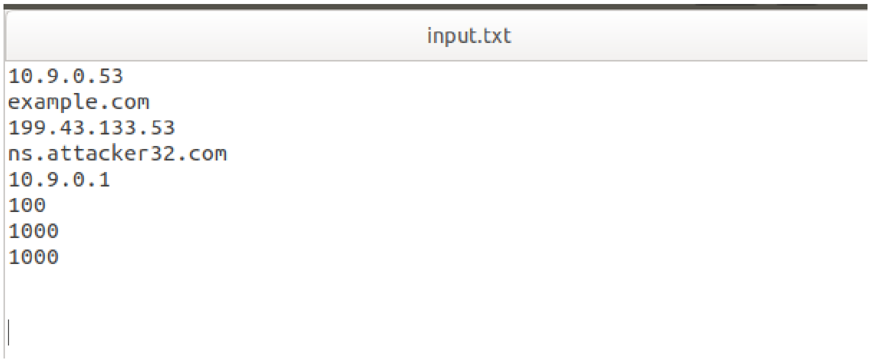
Attack Steps:

- I ran *rndc flush* command on the **local DNS server** machine.
- Then, I ran the attack code(1605071.c) on the **attacker**'s machine.

  The commands are:

  - *gcc 1605071.c -o <object-file-name>*
  - *./<object-file-name>*

  The inputs are:

  ```
  10.9.0.53
  example.com
  199.43.133.53
  ns.attacker32.com
  10.9.0.1
  100
  1000
  1000
  ```

  Here,

  10.9.0.53 = the ip of victim DNS

  example.com = the to be poisoned domain

  199.43.133.53 = the original IP of DNS server of example.com

  ns.attacker32.com = the attacker's nameserver

  10.9.0.1 = the attacker's ip

  100 = number of requests per subdomain

  1000 = number of responses per subdomain

  1000 = number of attack-attempts

Attack Justification:

- **Attackers can poison DNS caches by impersonating DNS nameservers**, making a request to a DNS resolver, and then pretending the response is from a legitimate server by forging the header data of the reply when the DNS resolver queries a nameserver. **This is possible because DNS servers use UDP** instead of TCP, and because currently there is no verification for DNS information.
- Most of the forged answers are dropped because the Query ID doesn't match, but **if just one in the flood of fake responses gets it right, the nameserver will accept the answer as genuine.** And because that satisfies the request, **the real answer that arrives later is dropped**, because the query is no longer pending.
- Here in this attack, even if the spoofed DNS response fails to poison the cache, **the attacker gets another chance to do the poisoning attack** as the next time the attacker will query a different name, the DNS server will send another query, giving the attacker another chance to guess the query ID correctly.
- **Probability calculation of the attack being successful:**

  For one attempt,

  the probability of being successful, s1 = (number_of_responses) / (2^16)
  the probability of being failed, f1 = 1 - s1
  = 1 - ((number_of_responses) / (2^16))

  For (number_of_attempts),

  the probability of being failed, f = (f1)^(number_of_attempts)
  = (1 - ((number_of_responses) / (2^16)))^(number_of_attempts)
  the probability of being successful, s = 1 - f
  = 1 - ((1 - ((number_of_responses) / (2^16)))^(number_of_attempts))

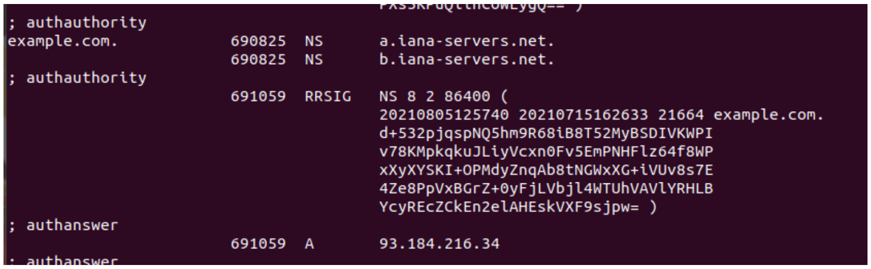
  Let, number_of_responses = 1000, number_of_attempts = 1000
  Then, The probability of being successful, s = 0.9999997901(which is close to 1)

Observed Output:

I ran the following two commands on the **local DNS server**:

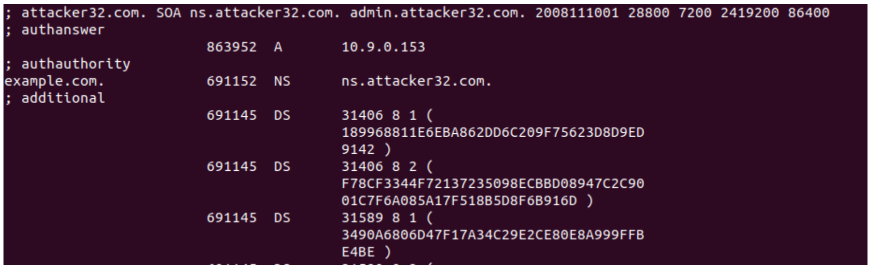
- *rndc dumpdb -cache*
- *nano dump.db*

Before Attack:(**dump.db** on **local DNS server**)



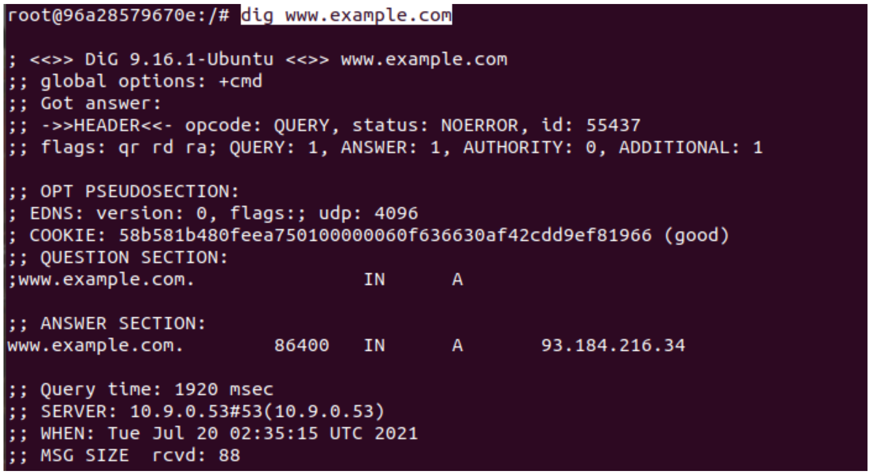
```
; authauthority
example.com.            690825  NS      a.iana-servers.net.
                        690825  NS      b.iana-servers.net.
; authauthority
                        691059  RRSIG   NS 8 2 86400 (
                                        20210805125740 20210715162633 21664 example.com.
                                        d+532pjqspNQ5hm9R68iB8T52MyBSDIVKWPI
                                        v78KMpkqkuJLiyVcxn0Fv5EmPNHFlz64f8WP
                                        xXyXYSKI+OPMdyZnqAb8tNGWxXG+iVUv8s7E
                                        4Ze8PpVxBGrZ+0yFjLVbjl4WTUhVAVlYRHLB
                                        YcyREcZCkEn2elAHEskVXF9sjpw= )
; authanswer
                        691059  A       93.184.216.34
; authanswer
```

After Attack:(**dump.db** on **local DNS server**)



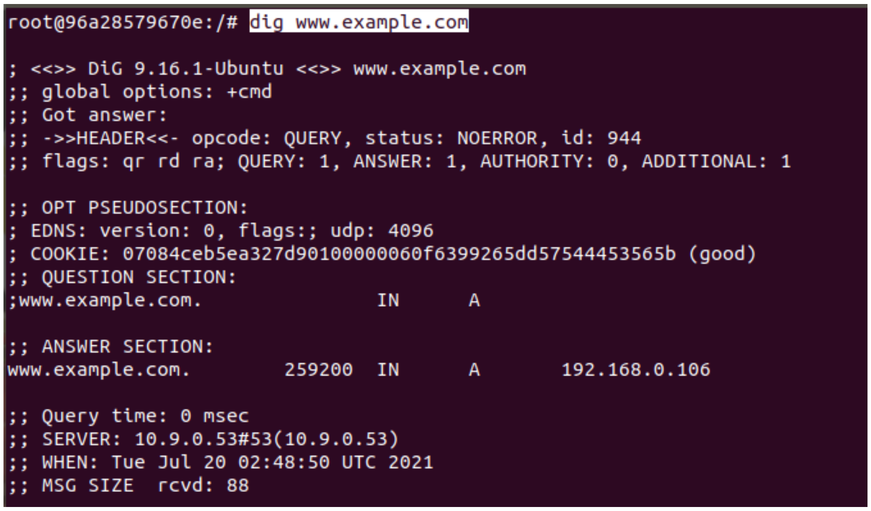
```
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
; authanswer
                        863952  A       10.9.0.153
; authauthority
example.com.            691152  NS      ns.attacker32.com.
; additional
                        691145  DS      31406 8 1 (
                                        189968811E6EBA862DD6C209F75623D8D9ED
                                        9142 )
                        691145  DS      31406 8 2 (
                                        F78CF3344F72137235098ECBBD08947C2C90
                                        01C7F6A085A17F518B5D8F6B916D )
                        691145  DS      31589 8 1 (
                                        3490A6806D47F17A34C29E2CE80E8A999FFB
                                        E4BE )
```

Before Attack:(On **user**’s machine)



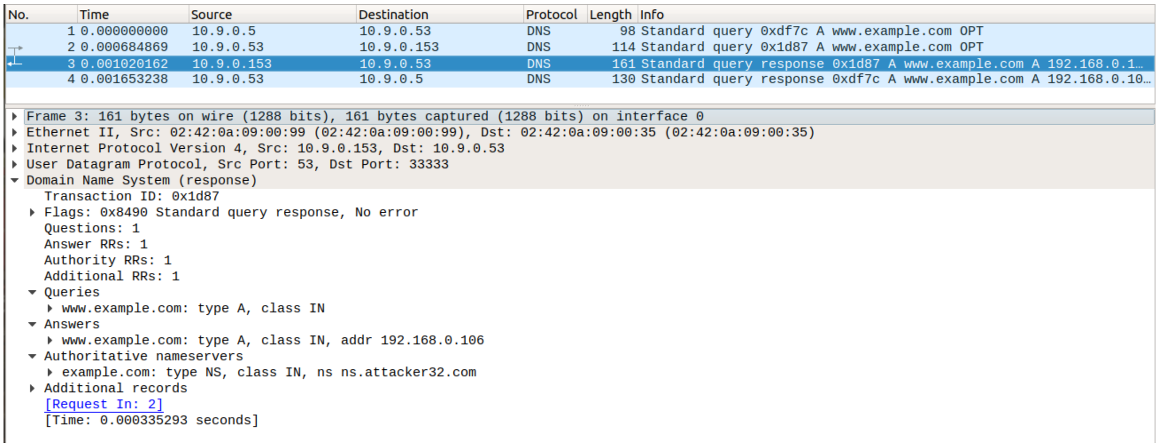
```
root@96a28579670e:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55437
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 58b581b480feea750100000060f636630af42cdd9ef81966 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 1920 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jul 20 02:35:15 UTC 2021
;; MSG SIZE  rcvd: 88
```

After Attack:(On **user**’s machine)



```
root@96a28579670e:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 944
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 07084ceb5ea327d90100000060f6399265dd57544453565b (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168.0.106

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jul 20 02:48:50 UTC 2021
;; MSG SIZE  rcvd: 88
```

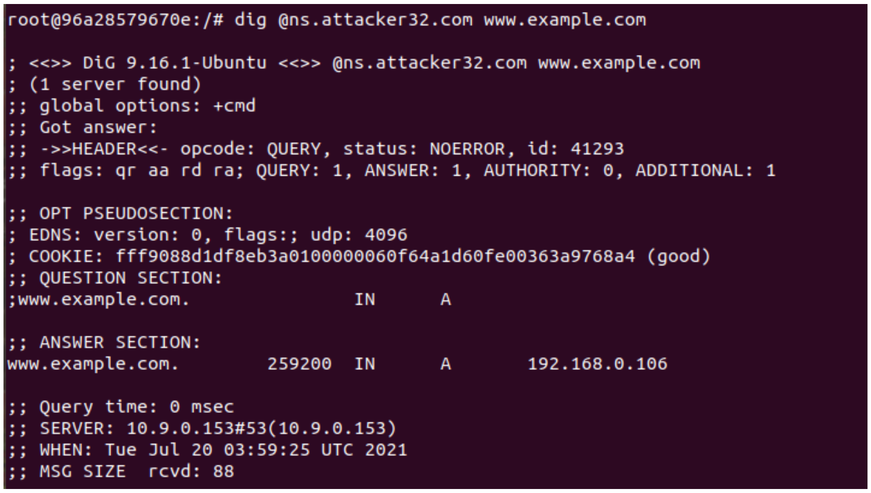
After Attack:(On **Wireshark**)



```
No.     Time          Source       Destination  Protocol Length Info
      1 0.000000000   10.9.0.5     10.9.0.53    DNS          98 Standard query 0xdf7c A www.example.com OPT
      2 0.000684869   10.9.0.53    10.9.0.153   DNS         114 Standard query 0x1d87 A www.example.com OPT
      3 0.001020162   10.9.0.153   10.9.0.53    DNS         161 Standard query response 0x1d87 A www.example.com A 192.168.0.1…
      4 0.001653238   10.9.0.53    10.9.0.5     DNS         130 Standard query response 0xdf7c A www.example.com A 192.168.0.10…

▸ Frame 3: 161 bytes on wire (1288 bits), 161 bytes captured (1288 bits) on interface 0
▸ Ethernet II, Src: 02:42:0a:09:00:99 (02:42:0a:09:00:99), Dst: 02:42:0a:09:00:35 (02:42:0a:09:00:35)
▸ Internet Protocol Version 4, Src: 10.9.0.153, Dst: 10.9.0.53
▸ User Datagram Protocol, Src Port: 53, Dst Port: 33333
▾ Domain Name System (response)
    Transaction ID: 0x1d87
  ▸ Flags: 0x8490 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 192.168.0.106
  ▾ Authoritative nameservers
    ▸ example.com: type NS, class IN, ns ns.attacker32.com
  ▸ Additional records
    [Request In: 2]
    [Time: 0.000335293 seconds]
```

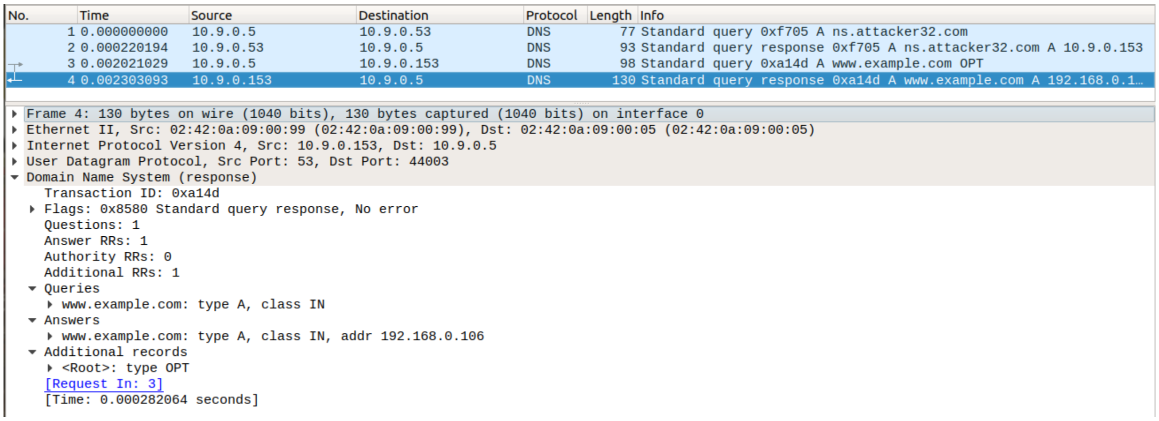
This indicates that the attack is successful. If ns.attacker32.com is used to query the domain, it gives a similar result.

(On **user**’s machine)



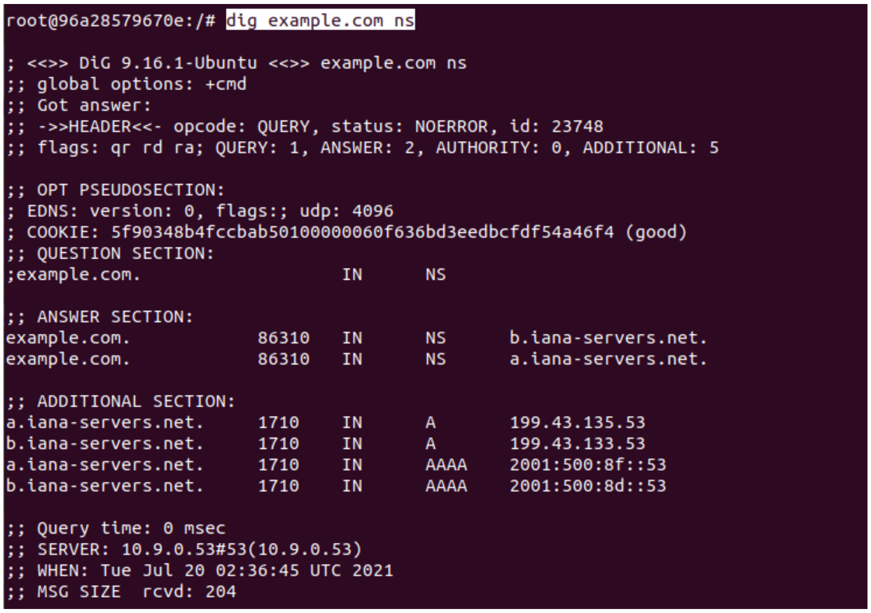
```
root@96a28579670e:/# dig @ns.attacker32.com www.example.com

; <<>> 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: 41293
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: fff9088d1df8eb3a0100000060f64a1d60fe00363a9768a4 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168.0.106

;; Query time: 0 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Jul 20 03:59:25 UTC 2021
;; MSG SIZE  rcvd: 88
```

(On **Wireshark**)



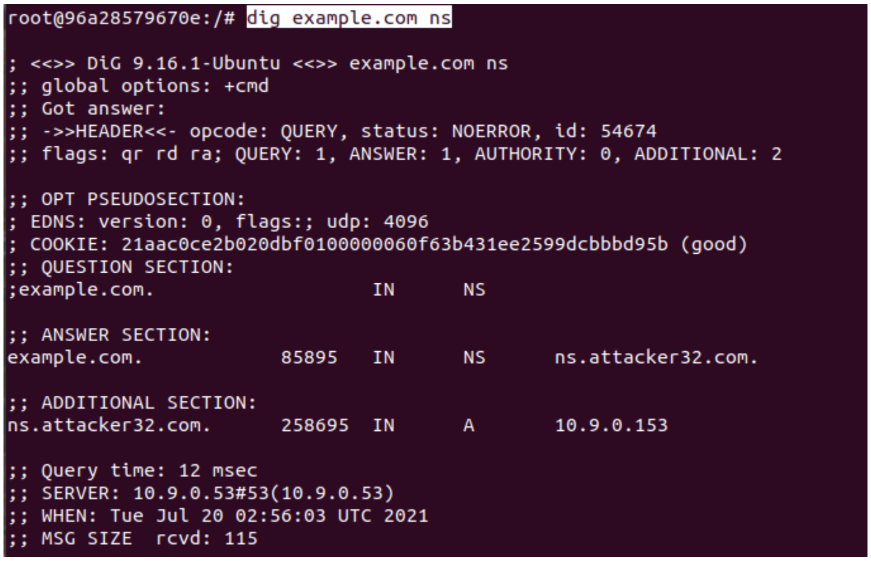
```
No.   Time          Source       Destination   Protocol  Length  Info
    1 0.000000000   10.9.0.5     10.9.0.53     DNS           77 Standard query 0xf705 A ns.attacker32.com
    2 0.000220194   10.9.0.53    10.9.0.5      DNS           93 Standard query response 0xf705 A ns.attacker32.com A 10.9.0.153
    3 0.002021029   10.9.0.5     10.9.0.153    DNS           98 Standard query 0xa14d A www.example.com OPT
    4 0.002303093   10.9.0.153   10.9.0.5      DNS          130 Standard query response 0xa14d A www.example.com A 192.168.0.1…

▸ Frame 4: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
▸ Ethernet II, Src: 02:42:0a:09:00:99 (02:42:0a:09:00:99), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
▸ Internet Protocol Version 4, Src: 10.9.0.153, Dst: 10.9.0.5
▸ User Datagram Protocol, Src Port: 53, Dst Port: 44003
▾ Domain Name System (response)
    Transaction ID: 0xa14d
  ▸ Flags: 0x8580 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 1
  ▾ Queries
    ▸ www.example.com: type A, class IN
  ▾ Answers
    ▸ www.example.com: type A, class IN, addr 192.168.0.106
  ▾ Additional records
    ▸ <Root>: type OPT
    [Request In: 3]
    [Time: 0.000282064 seconds]
```

Before Attack:(On **user**'s machine)



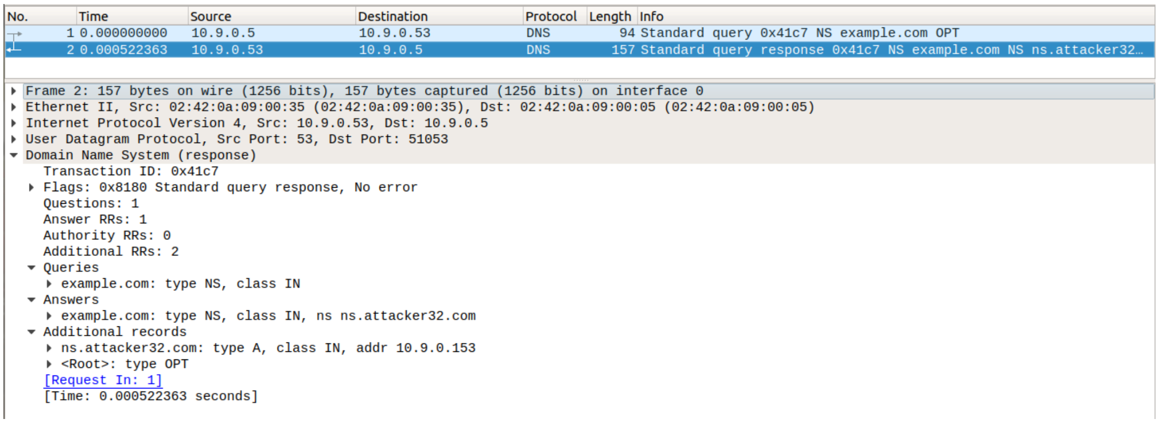
```
root@96a28579670e:/# dig example.com ns

; <<>> DiG 9.16.1-Ubuntu <<>> example.com ns
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23748
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5f90348b4fccbab50100000060f636bd3eedbcfdf54a46f4 (good)
;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            86310   IN      NS      b.iana-servers.net.
example.com.            86310   IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     1710    IN      A       199.43.135.53
b.iana-servers.net.     1710    IN      A       199.43.133.53
a.iana-servers.net.     1710    IN      AAAA    2001:500:8f::53
b.iana-servers.net.     1710    IN      AAAA    2001:500:8d::53

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jul 20 02:36:45 UTC 2021
;; MSG SIZE  rcvd: 204
```

After Attack:(On **user**'s machine)



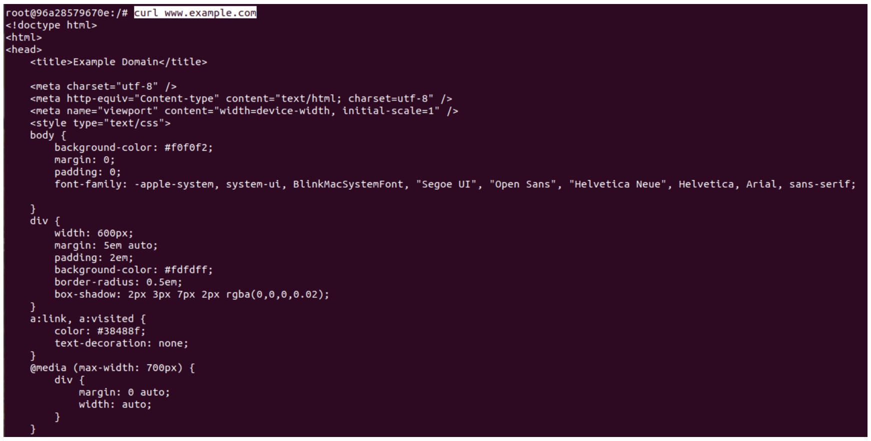
```
root@96a28579670e:/# dig example.com ns

; <<>> DiG 9.16.1-Ubuntu <<>> example.com ns
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54674
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 21aac0ce2b020dbf0100000060f63b431ee2599dcbbbd95b (good)
;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            85895   IN      NS      ns.attacker32.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      258695  IN      A       10.9.0.153

;; Query time: 12 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jul 20 02:56:03 UTC 2021
;; MSG SIZE  rcvd: 115
```

After Attack:(On **Wireshark**)



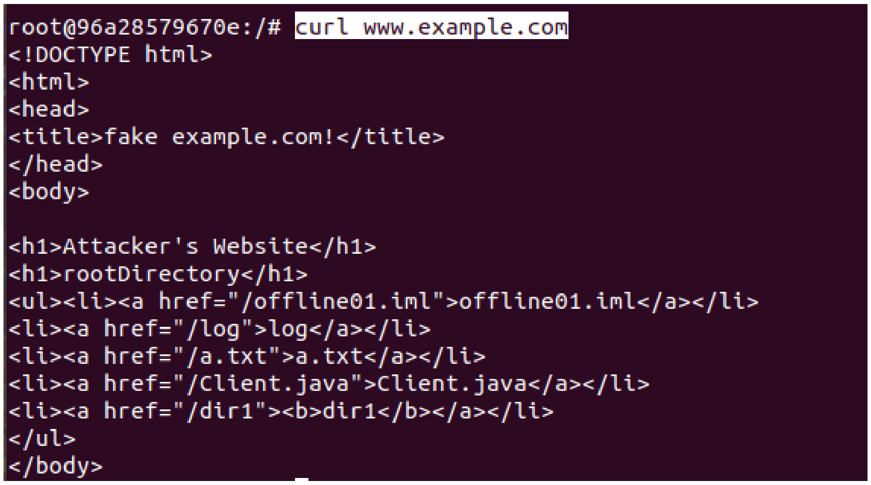
```
No.   Time           Source       Destination   Protocol  Length  Info
    1 0.000000000    10.9.0.5     10.9.0.53     DNS           94  Standard query 0x41c7 NS example.com OPT
    2 0.000522363    10.9.0.53    10.9.0.5      DNS          157  Standard query response 0x41c7 NS example.com NS ns.attacker32…

▸ Frame 2: 157 bytes on wire (1256 bits), 157 bytes captured (1256 bits) on interface 0
▸ Ethernet II, Src: 02:42:0a:09:00:35 (02:42:0a:09:00:35), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
▸ Internet Protocol Version 4, Src: 10.9.0.53, Dst: 10.9.0.5
▸ User Datagram Protocol, Src Port: 53, Dst Port: 51053
▾ Domain Name System (response)
    Transaction ID: 0x41c7
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 2
  ▾ Queries
    ▸ example.com: type NS, class IN
  ▾ Answers
    ▸ example.com: type NS, class IN, ns ns.attacker32.com
  ▾ Additional records
    ▸ ns.attacker32.com: type A, class IN, addr 10.9.0.153
    ▸ <Root>: type OPT
    [Request In: 1]
    [Time: 0.000522363 seconds]
```

Before Attack:(On **user**’s machine)



```
root@96a28579670e:/# curl www.example.com
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
```

After Attack:(On **user**’s machine)



```
root@96a28579670e:/# curl www.example.com
<!DOCTYPE html>
<html>
<head>
<title>fake example.com!</title>
</head>
<body>

<h1>Attacker's Website</h1>
<h1>rootDirectory</h1>
<ul><li><a href="/offline01.iml">offline01.iml</a></li>
<li><a href="/log">log</a></li>
<li><a href="/a.txt">a.txt</a></li>
<li><a href="/Client.java">Client.java</a></li>
<li><a href="/dir1"><b>dir1</b></a></li>
</ul>
</body>
```

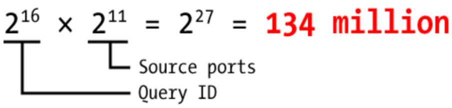
CounterMeasure:

- **Relying less on other DNS:**
    - The DNS servers should be configured by an IT professional to rely very little on relationships with other DNS servers. This makes it much harder for cyber-criminal to use their DNS server to corrupt their targets, meaning the DNS server is less likely to be corrupted.
    - DNS servers should be configured to rely as little as possible on trust relationships with other DNS servers. Configuring it this way will make it much more difficult for an attacker to use their own DNS server to corrupt a targeted server.
- **Running only required services:**

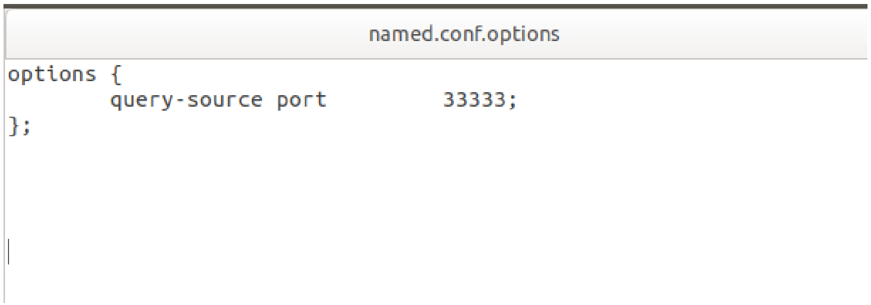
  DNS servers should be set up so that only services that are required are ones that are allowed to run. Having additional services that are not required running on a DNS server just increases the attack vector size.
- **Most recent version of DNS:**

  The most recent version of the DNS should be utilized. This is because the most recent versions will use security features such as
    - **port randomization** and
    - **transaction IDs that are cryptographically secure** to help guard against poisoning attacks.

  Port randomization increases the search space from 64k(2^16) to 134 million.



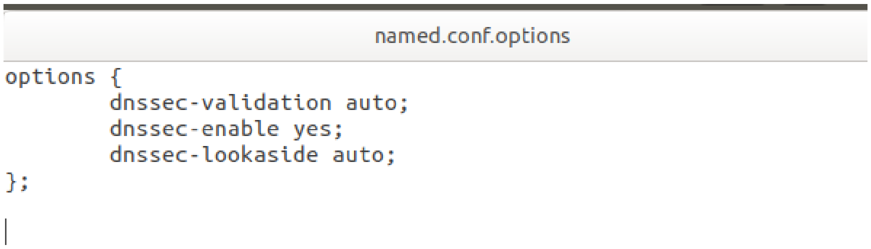
In this project, **port randomization** can be enabled by removing the following snippet from the file **named.conf.options** in the **local DNS server**:



```
options {
        query-source port        33333;
};
```

- **DNSSEC**(Domain Name System Security Extension):

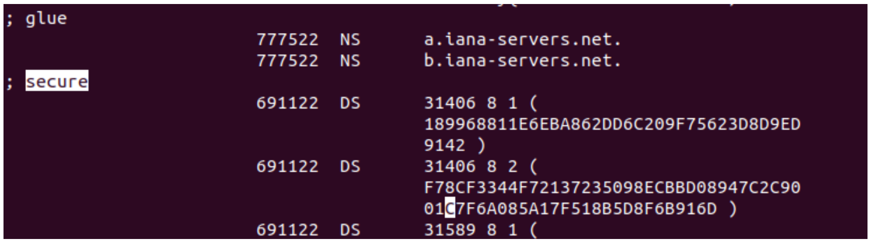
  In this project, **DNSSEC** can be enabled by editing the file **named.conf.options** in the local **local DNS server**.



named.conf.options

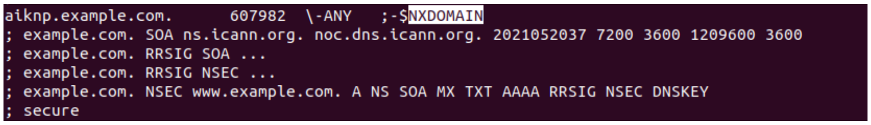
```
options {
        dnssec-validation auto;
        dnssec-enable yes;
        dnssec-lookaside auto;
};
```

  - When it is deployed, computers are able to confirm if DNS responses are legitimate.



```
; glue
                        777522  NS      a.iana-servers.net.
                        777522  NS      b.iana-servers.net.
; secure
                        691122  DS      31406 8 1 (
                                        189968811E6EBA862DD6C209F75623D8D9ED
                                        9142 )
                        691122  DS      31406 8 2 (
                                        F78CF3344F72137235098ECBBD08947C2C90
                                        01C7F6A085A17F518B5D8F6B916D )
                        691122  DS      31589 8 1 (
```

  - It also has the ability to verify that a domain name does not exist at all, which can help prevent man-in-the-middle attacks.



```
aiknp.example.com.      607982  \-ANY   ;-$NXDOMAIN
; example.com. SOA ns.icann.org. noc.dns.icann.org. 2021052037 7200 3600 1209600 3600
; example.com. RRSIG SOA ...
; example.com. RRSIG NSEC ...
; example.com. NSEC www.example.com. A NS SOA MX TXT AAAA RRSIG NSEC DNSKEY
; secure
```

    (An **NXDOMAIN** error message means that the domain does not exist.)