Concept of DNS and Interacting with a DNS Server

Mohammad Shahadat Hossain

PMICS|CSE|DU

Lab-1 – DNS Concept, Implementation

- Concept of DNS and Name Resolution
- Role of DNS in Internet communication
- Structure of World-Wide DNS Implementation
- Types and Role of DNS Sever
- DNS Security Issues and Solutions
  - Zone Transfer
  - DNS Cache Poisoning
- Performing different type of DNS query for response
- Analysis of DNS query and response packets
- Explaining DNS query and response packet headers

DNS Basics

What is DNS Protocol

The DNS (Domain Name System) Protocol is a standardized communication protocol used to resolve human-readable domain names (like www.google.com) into IP addresses (like 142.250.190.4) that computers use to identify each other on the Internet. It's a hierarchical and distributed naming system.

How DNS Protocol Works

1. User types a domain name in a browser.
2. The browser asks the OS to resolve it.
3. OS asks the configured DNS server.
4. The DNS server uses the DNS protocol (usually over UDP port 53) to:
   a. Check its cache.
   b. Query other DNS servers (root, TLD, authoritative) recursively.
5. It returns the final IP address to the OS/browser.
6. The browser connects to the server using the IP address.

Protocol Characteristics

- Uses **UDP port 53** for most queries.
- Uses **TCP port 53** for larger data (e.g., zone transfers).
- Now supports **secure extensions** like:
  - **DNSSEC** – authentication and integrity.
  - **DoT** (DNS over TLS), **DoH** (DNS over HTTPS) – encryption.

Role of DNS in Internet World

The primary role of the Domain Name System (DNS) is to act as the internet's "phonebook," translating human-friendly domain names like www.google.com into computer-friendly IP addresses, enabling effortless navigation of the web. Without DNS, users would have to memorize long strings of numbers to access websites and online services, making it an essential and ubiquitous infrastructure for almost every internet transaction.

Name-to-IP resolution:

- When you type a URL into your browser, your computer sends a query to a DNS resolver to find the corresponding IP address.

Hierarchical lookup:

- The resolver may query other DNS servers in a hierarchical system—from root servers to top-level domain (TLD) servers and finally to the authoritative nameserver for the specific domain—to find the IP address.

Response to the client:

- Once the IP address is found, the DNS server sends it back to your computer, which then uses the IP address to establish a connection with the web server and load the page.

Role of DNS in Internet World

Key functions and benefits

Simplifies navigation:

- DNS makes the internet easy to use by allowing people to use memorable names instead of numerical IP addresses.

Supports various applications:

- It is critical for more than just web browsing; it also enables email addresses and other online services to function correctly.

Ensures scalability and robustness:

- The distributed nature of the DNS system, with its worldwide network of servers, makes it robust and scalable.

Increases performance:

- DNS caching stores frequently accessed IP addresses locally on your computer or by your ISP, which speeds up future visits to the same websites by bypassing the full lookup process.

Provides security:

- Modern security protocols like DNS over HTTPS (DoH) and DNS over TLS (DoT) encrypt DNS queries to protect against eavesdropping and manipulation.

DNS Server and Roles

A DNS Server is a computer that implements the DNS protocol. It listens to DNS requests and responds based on its role.

Types of DNS Servers:

Server Type	Role
Recursive Resolver	Accepts queries from clients, resolves names on their behalf. Caches results.
Root Server	First step in the resolution process. Directs queries to TLD servers.
TLD Server	Points to authoritative servers for domains like .com, .org, .bd.
Authoritative Server	Contains actual DNS records (A, MX, TXT) for a domain. Returns the final answer.
Forwarder	Relays queries to another DNS server. Common in enterprise networks.
Caching Server	Stores responses temporarily to speed up future queries.

DNS Server and Roles (Root DNS)

- **Role**: The top of the DNS hierarchy; they don't store specific domain records but know where to find information for each **Top-Level Domain (TLD)**.

- **Function**:
  - Respond to queries for domain names with a list of authoritative servers for the relevant TLD (like .com, .org, .bd).
  - Example: When you query for www.example.com, the root server tells your resolver which .com TLD server to ask next.

- **Count**: 13 root server clusters (A–M), each replicated globally for performance and reliability.

DNS Server and Roles (TLD Name Servers)

Role: Manage information for top-level domains (TLDs), such as .com, .org, .net, .bd, etc.

Function:

Store and serve authoritative data about second-level domains within the TLD.

Example: The .com TLD servers know which authoritative server handles example.com.

Example TLD Operators:

.com — managed by Verisign

.org — managed by Public Interest Registry

.bd — managed by Bangladesh Telecommunications Company Limited (BTCL)

DNS Server and Roles (Authoritative Name Servers)

Role: Contain the definitive DNS records for a domain (like A, MX, CNAME, TXT records).

Function:

- Answer DNS queries about domains they host.
- Do **not** perform recursive lookups; they directly provide answers from stored zone files.

Example:

An authoritative DNS server for example.com returns:

A record → 192.168.1.10
MX record → mail.example.com

Types of Authoritative Servers:

Primary (Master): Maintains the original zone files.
Secondary (Slave): Gets copies of the zone files via **Zone Transfer (AXFR/IXFR)** for redundancy.

DNS Server and Roles (Forwarding Name Servers)

Role: Forwards DNS queries to another DNS server (usually a recursive resolver) instead of resolving them directly.

Function:

- Acts as an intermediary for internal networks.
- Often used to control and log DNS traffic or apply filtering policies.

Example:

- A corporate DNS server forwards external domain queries to Google DNS but resolves internal names locally.

DNS Server and Roles (Caching-Only Name Servers)

Role: Do not host any domain zone; purely store results of previous lookups.

Function:

- Improves query performance by reducing the need to repeatedly query the root/TLD servers.
- Clears entries once the **TTL (Time to Live)** expires.

DNS Server and Roles (Caching-Only Name Servers)

DNS Server Type	Primary Role	Key Function
Root Name Server	Top-level DNS reference	Directs queries to TLD servers
TLD Server	Manages domain extensions	Points to authoritative servers
Authoritative Server	Final source of truth	Stores domain's actual records
Recursive Resolver	Full query resolver	Queries DNS hierarchy on behalf of clients
Forwarding Server	Forwards queries	Applies caching or policy control
Caching-only Server	Performance booster	Stores recent lookups only

DNS Query Resolution Flow (Recursive Query)

In a recursive query, the DNS client asks a DNS server to **resolve the full domain name** and return the **final answer** — either the IP address or an error (if the domain doesn't exist).

How It Works:

1. A client (e.g., your browser) asks its local DNS resolver:
   "What is the IP address of www.example.com?"
2. The resolver takes full responsibility to find the answer.
   - It queries the **root server**, then the **TLD server**, then the **authoritative server**.
   - Once it finds the IP address, it returns it to the client.
3. The client does not interact with other servers; only the resolver does.

Example Flow:

Client → Resolver: "What is **www.example.com**?"
Resolver → Root → TLD → Authoritative → Returns IP
Resolver → Client: "192.0.2.10"

Used By:

- End-user devices (browsers, PCs etc...)
- Recursive DNS resolvers (like Google's 8.8.8.8 or Cloudflare's 1.1.1.1)

Advantages:

- Simple for the client; it only sends one query.
- Reduces network complexity for end-users.

Disadvantages:

- Heavy load on the recursive resolver.
- Vulnerable to DNS cache poisoning if not secured.

DNS Query Resolution Flow (Iterative Query)

In an iterative query, the DNS client (usually a resolver) asks a DNS server for information, and if the server doesn't know the final answer, it replies with a **referral** — the address of another DNS server that might know.

How It Works:

- The resolver asks the **root server** for www.example.com.
- The root server replies: "Ask the .com TLD server."
- The resolver asks the **.com TLD server**, which says: "Ask the authoritative server for example.com."
- Finally, the resolver asks the **authoritative server**, which replies with the IP address.
- The resolver then gives the final answer to the client.

Example Flow:

Resolver → Root: "www.example.com?"
Root → Resolver: "Ask .com TLD at 192.0.34.166"
Resolver → TLD: "www.example.com?"
TLD → Resolver: "Ask authoritative server at 203.0.113.53"
Resolver → Authoritative: "www.example.com?"
Authoritative → Resolver: "192.0.2.10"

Used By:

DNS servers communicating with each other (root → TLD → authoritative)
Recursive resolvers when resolving queries for clients.

Advantages:

- Efficient for DNS infrastructure.
- Reduces load on higher-level DNS servers (root/TLD).

Disadvantages:

- The resolver must perform multiple queries.
- Slower if not cached.

DNS Query Resolution Flow Comparison

Feature	Recursive Query	Iterative Query
Responsibility	DNS server must find full answer	DNS server gives referral if it doesn't know
Query Chain	Client → Resolver (only one interaction)	Resolver → Multiple servers (Root, TLD, Authoritative)
Used By	Clients and end devices	DNS resolvers and servers
Complexity for Client	Simple (one query)	Complex (resolver handles multiple queries)
Load on Server	Higher on resolver	Distributed across servers
Response Type	Final answer or error	Referral or final answer
Example User	Browser querying ISP DNS	Resolver querying Root/TLD servers

Simple Analogy

Imagine asking for a person's address:

- **Recursive Query:** You ask your friend to find the address. He goes, asks around, and comes back with the final answer.
- **Iterative Query:** You ask one person; they say, "Ask the office down the street." You go there; they say, "Ask the next building." You continue until you get the answer.

DNS diagnostic tool: dig

The dig command in Linux is a DNS (Domain Name System) tool used for querying DNS servers to retrieve information about domain names. Its primary usage is to resolve domain names to IP addresses or retrieve various DNS records like A (IPv4 address), MX (Mail exchange), CNAME (Canonical name), and more.

dig utilizes the operating system's resolver libraries, making it more reliable and consistent across different systems.

Syntax

The basic syntax of the dig command:

```
dig  [OPTIONS]  [NAME/IP]  [TYPE]
```

NAME: The domain name or IP address you want to query.

TYPE: The type of DNS record you want to query. E.G. A, NS, MX, PTR, CNAME etc.....

DNS diagnostic tool: dig command OPTIONS

OPTIONS are optional, specifying at least the NAME and TYPE is necessary for most dig queries:
+short: Provides concise output.
+trace: Traces the DNS query process from root servers to authoritative servers for detailed resolution path information.
+stats: Displays query statistics, including query time and response size.
@server: Specify a specific DNS server to query.
-x: Performs a reverse DNS lookup based on an IP address.
+tcp: Forces the use of TCP for the DNS query, suitable for large responses or when UDP is blocked.
+class: Specify the DNS class for the query (default is "IN").
+comments: Displays comments in DNS responses for additional information.
+question: Shows the question section of the DNS response.
+answer: Displays the answer section of the DNS response.

+authority: Shows the authority section of the DNS response.

+additional: Displays additional information in the DNS response.

+**noquestion, +noanswer, +noauthority, +noadditional**: Hide specific sections in the DNS response.

+nocl: Hides the class in DNS records.

+nocmd: Hides command information in the output.

+nocomments: Hides comments in DNS responses.

+nostats: Hides query statistics.

+noall: Hides all sections except for the answer section.

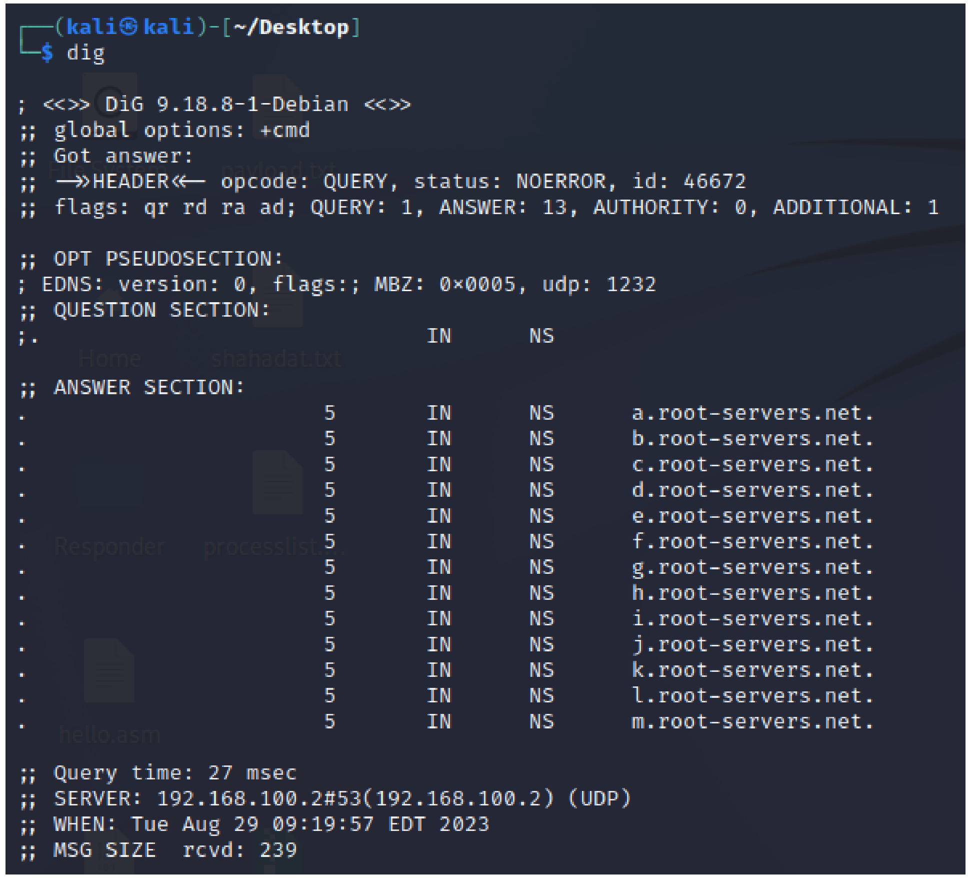
Key DNS Flags

Key DNS Header Flags:

- **QR: Indicates if the message is a query (0) or a response (1).**
- **Opcode:** Specifies the type of query (e.g., standard, update, NS).
- **AA:** Indicates if the response is an authoritative answer from a name server.
- **TC:** Specifies if the message is truncated.
- **RD: Indicates if recursion is desired by the client.**
- **RA:** Indicates if recursion is available on the server.
- **AD:** Indicates if the response has been verified by the server using DNSSEC.
- **CD:** Specifies if the client is willing to accept non-verified data (DNSSEC).

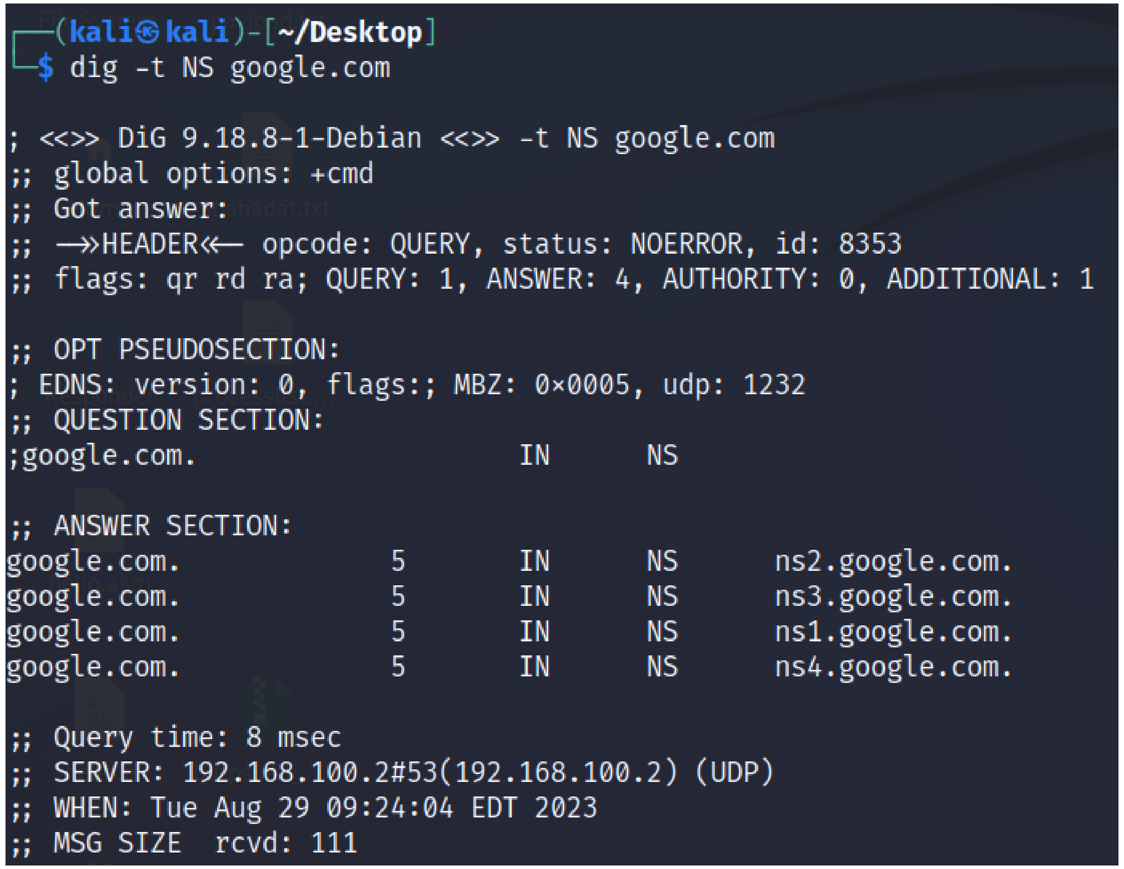
Communicating with DNS

Know the root servers



```
┌──(kali㉿kali)-[~/Desktop]
└─$ dig

; <<>> DiG 9.18.8-1-Debian <<>>
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46672
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 0×0005, udp: 1232
;; QUESTION SECTION:
;.                              IN      NS

;; ANSWER SECTION:
.                       5       IN      NS      a.root-servers.net.
.                       5       IN      NS      b.root-servers.net.
.                       5       IN      NS      c.root-servers.net.
.                       5       IN      NS      d.root-servers.net.
.                       5       IN      NS      e.root-servers.net.
.                       5       IN      NS      f.root-servers.net.
.                       5       IN      NS      g.root-servers.net.
.                       5       IN      NS      h.root-servers.net.
.                       5       IN      NS      i.root-servers.net.
.                       5       IN      NS      j.root-servers.net.
.                       5       IN      NS      k.root-servers.net.
.                       5       IN      NS      l.root-servers.net.
.                       5       IN      NS      m.root-servers.net.

;; Query time: 27 msec
;; SERVER: 192.168.100.2#53(192.168.100.2) (UDP)
;; WHEN: Tue Aug 29 09:19:57 EDT 2023
;; MSG SIZE  rcvd: 239
```

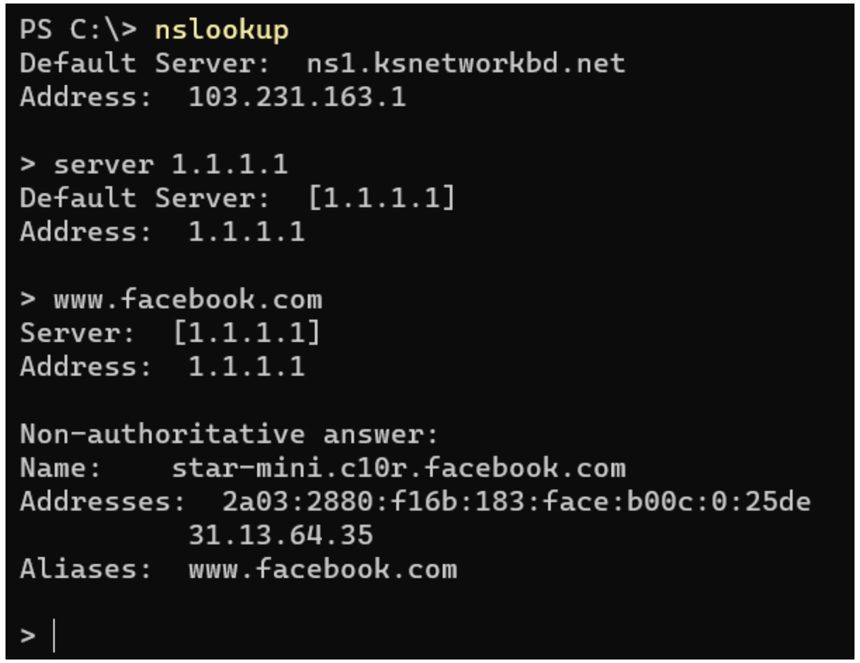
Verify the Name Server Records



```
┌──(kali㉿kali)-[~/Desktop]
└─$ dig -t NS google.com

; <<>> DiG 9.18.8-1-Debian <<>> -t NS google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8353
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 0×0005, udp: 1232
;; QUESTION SECTION:
;google.com.                    IN      NS

;; ANSWER SECTION:
google.com.             5       IN      NS      ns2.google.com.
google.com.             5       IN      NS      ns3.google.com.
google.com.             5       IN      NS      ns1.google.com.
google.com.             5       IN      NS      ns4.google.com.

;; Query time: 8 msec
;; SERVER: 192.168.100.2#53(192.168.100.2) (UDP)
;; WHEN: Tue Aug 29 09:24:04 EDT 2023
;; MSG SIZE  rcvd: 111
```

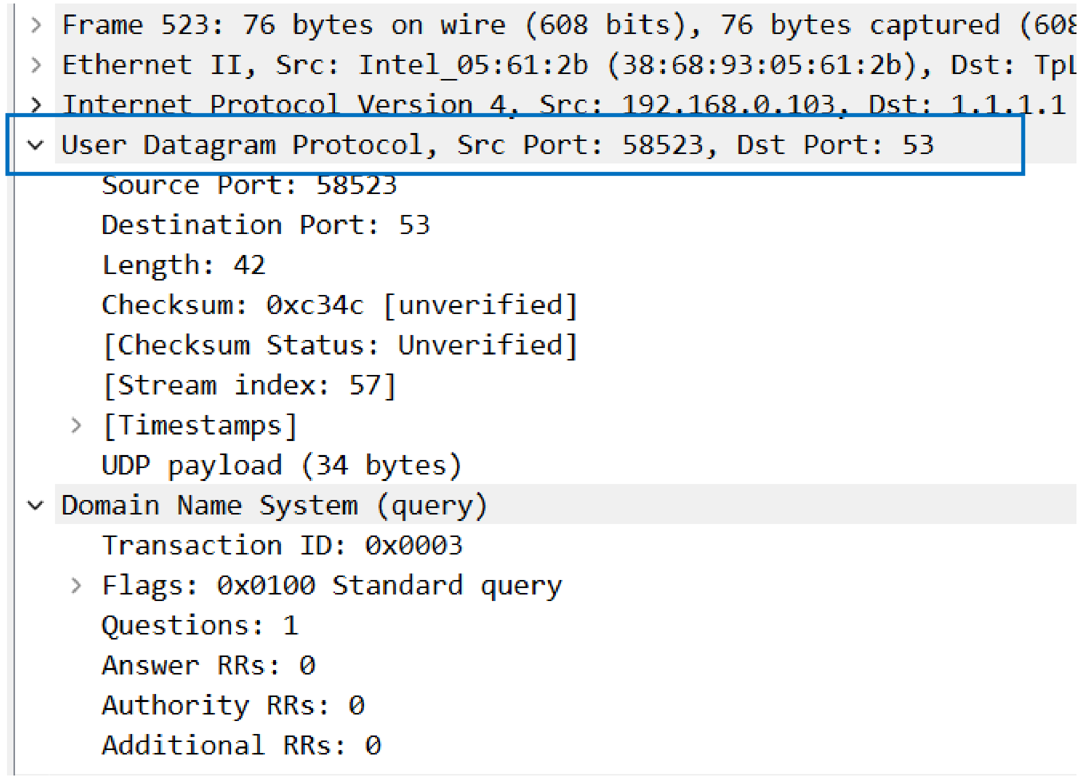
Communicating with DNS

Verify the Transport Layer Protocol used for Name Resolution



```
PS C:\> nslookup
Default Server:  ns1.ksnetworkbd.net
Address:  103.231.163.1

> server 1.1.1.1
Default Server:  [1.1.1.1]
Address:  1.1.1.1

> www.facebook.com
Server:  [1.1.1.1]
Address:  1.1.1.1

Non-authoritative answer:
Name:    star-mini.c10r.facebook.com
Addresses:  2a03:2880:f16b:183:face:b00c:0:25de
          31.13.64.35
Aliases:  www.facebook.com

> |
```

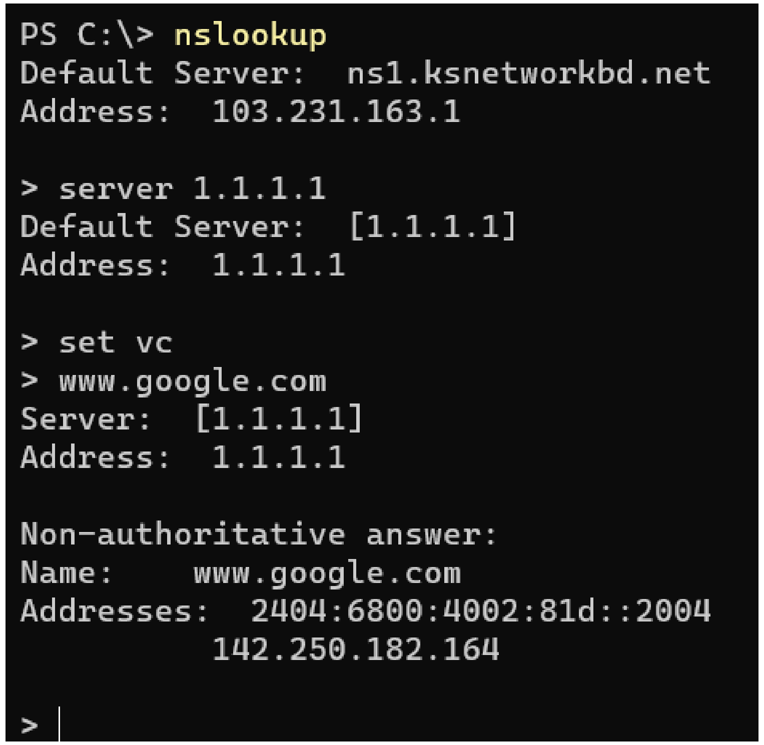
DNS Transport Layer Packet Analysis



```
> Frame 523: 76 bytes on wire (608 bits), 76 bytes captured (608
> Ethernet II, Src: Intel_05:61:2b (38:68:93:05:61:2b), Dst: Tpl
> Internet Protocol Version 4, Src: 192.168.0.103, Dst: 1.1.1.1
v User Datagram Protocol, Src Port: 58523, Dst Port: 53
    Source Port: 58523
    Destination Port: 53
    Length: 42
    Checksum: 0xc34c [unverified]
    [Checksum Status: Unverified]
    [Stream index: 57]
  > [Timestamps]
    UDP payload (34 bytes)
v Domain Name System (query)
    Transaction ID: 0x0003
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
```

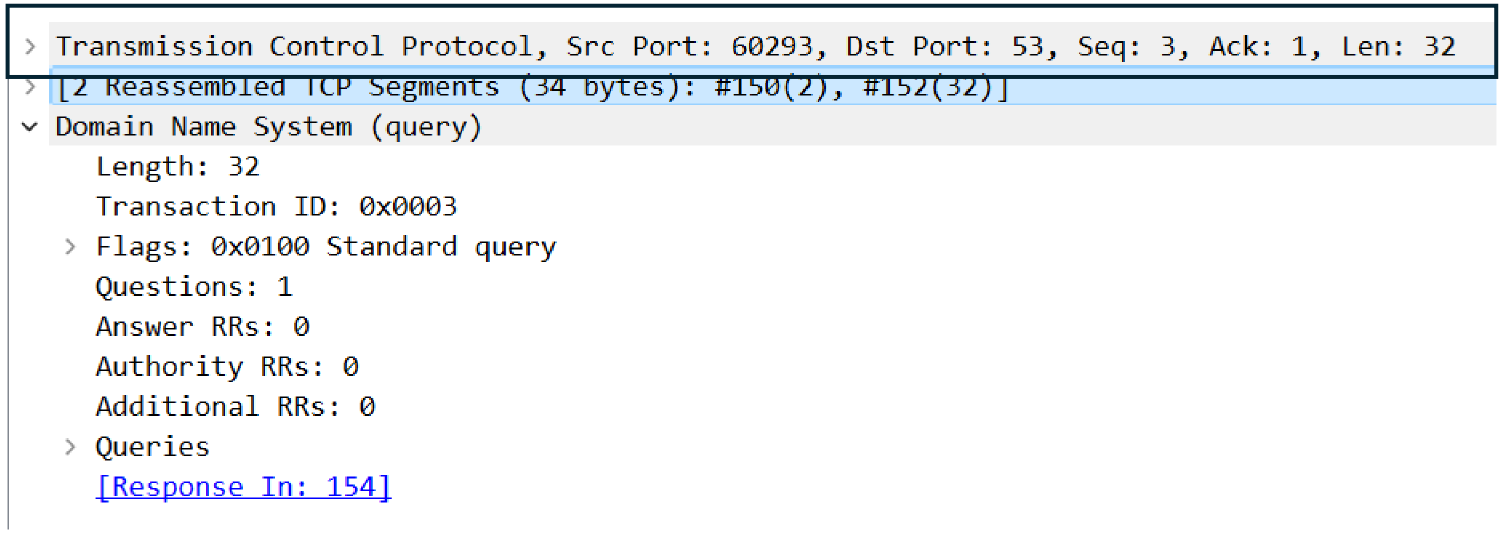
Communicating with DNS

Change the Transport Layer Protocol for Name Resolution



```
PS C:\> nslookup
Default Server:  ns1.ksnetworkbd.net
Address:  103.231.163.1

> server 1.1.1.1
Default Server:  [1.1.1.1]
Address:  1.1.1.1

> set vc
> www.google.com
Server:  [1.1.1.1]
Address:  1.1.1.1

Non-authoritative answer:
Name:    www.google.com
Addresses:  2404:6800:4002:81d::2004
          142.250.182.164

>
```

DNS Transport Layer Packet Analysis



```
> Transmission Control Protocol, Src Port: 60293, Dst Port: 53, Seq: 3, Ack: 1, Len: 32
> [2 Reassembled TCP Segments (34 bytes): #150(2), #152(32)]
v Domain Name System (query)
    Length: 32
    Transaction ID: 0x0003
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  > Queries
    [Response In: 154]
```

DNS Zone Transfer Concept

A **DNS zone transfer** is a mechanism used to **replicate DNS database records** from a **Primary (Master)** DNS server to one or more **Secondary (Slave)** DNS servers.

This ensures **redundancy**, **load balancing**, and **fault tolerance** across DNS infrastructure.

Why Zone Transfers Are Needed

- To **synchronize** DNS records between multiple DNS servers.
- To ensure **continuity of service** if one DNS server fails.
- To improve **query response performance** in geographically distributed networks.

DNS Zone Transfer Concept

Type	Name	Description
AXFR	Full Zone Transfer	Transfers the entire DNS zone file from the master to the slave server.
IXFR	Incremental Zone Transfer	Transfers only the **changes (deltas)** since the last synchronization — faster and efficient.

Example command (for testing):

```
dig @ns1.example.com example.com AXFR
```

How It Works

The **secondary DNS server** queries the **primary DNS server** for the zone's **SOA (Start of Authority)** record.

- It compares the **serial number** in the SOA record to check for updates.
- If the serial number is higher on the master, it initiates a zone transfer (AXFR or IXFR).
- The zone file is copied, and the secondary server updates its records.

Security Risks of Zone Transfers

Information Disclosure

If zone transfers are **not restricted**, anyone can request a full copy of your DNS zone file.
That reveals:

- Internal hostnames (e.g., mail, db, intranet)
- IP addresses (public and private)
- Network structure and relationships
- Third-party connections

Example Impact:

Attackers can map your organization’s internal infrastructure and plan targeted attacks.

Security Risks Mitigation Strategies

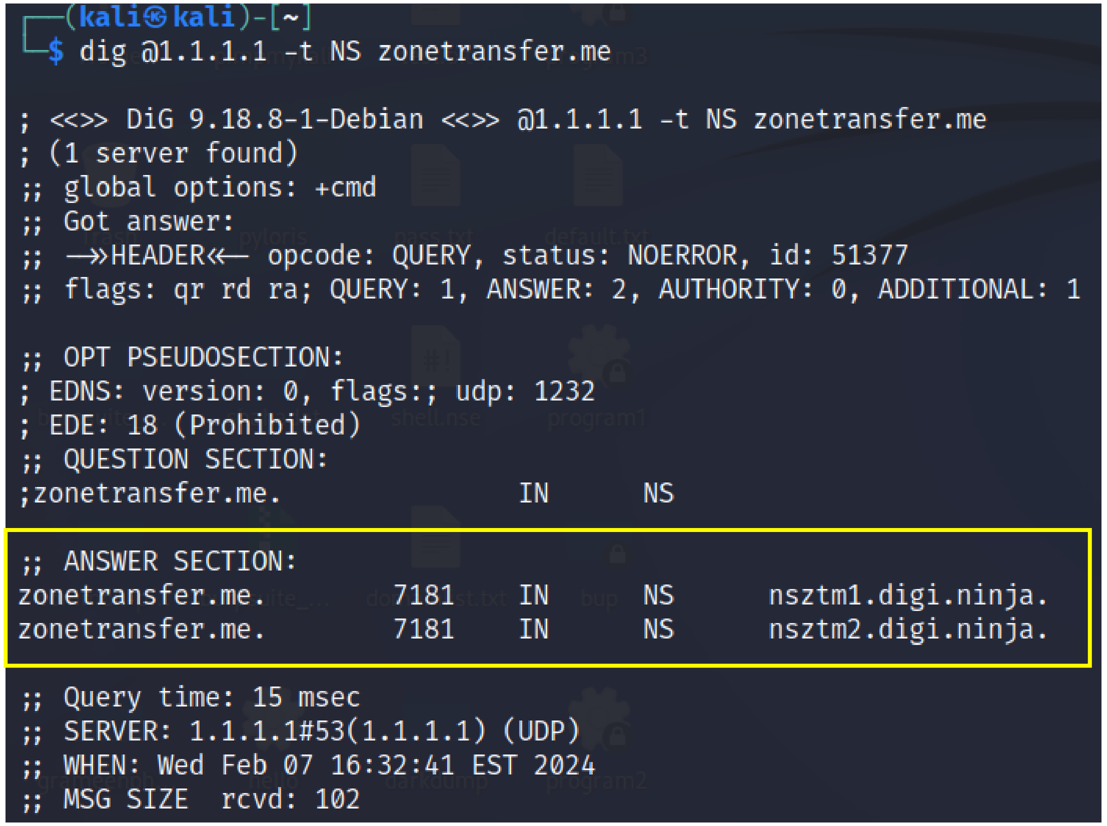
Control	Purpose	Implementation Example
Restrict Zone Transfers	Limit AXFR/IXFR to authorized IPs only.	In BIND: allow-transfer { 192.0.2.10; };
Use TSIG (Transaction Signatures)	Authenticate transfers using shared secret keys.	Configure with hmac-sha256 TSIG keys.
Monitor DNS Logs	Detect unauthorized AXFR attempts.	Review /var/log/named/ or SIEM alerts.
Disable Zone Transfers if Unused	Reduce attack surface.	Set allow-transfer { none; };
Implement Split DNS	Separate internal vs. external DNS zones.	Internal DNS only visible inside network.
Network Segmentation	Place DNS servers in DMZs with limited access.	Use firewalls to restrict inbound/outbound zone traffic.

Verifying the zone transfer configuration

- We will use **dig** command in this lab session
- Select the domain name for which we want to transfer the zone information. Our case it is **zonetransfer.me.** We resolve the name using DNS server 1.1.1.1

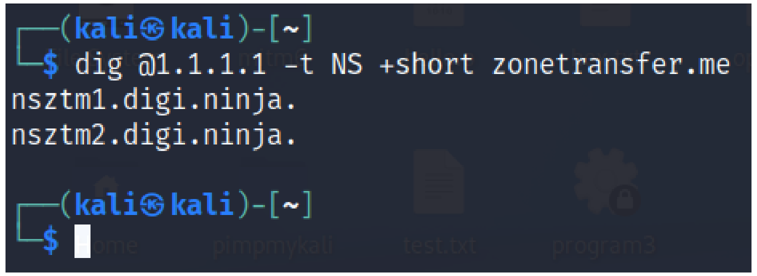
Identify the DNS server name of the domain zonetransfer.me

- Find the DNS server name of the domain using the **$dig –t NS @1.1.1.1 zonetransfer.me** command
- To get the result precisely you can use **$dig –t NS +short @1.1.1.1 zonetransfer.me** command



```
┌──(kali㉿kali)-[~]
└─$ dig @1.1.1.1 -t NS zonetransfer.me

; <<>> DiG 9.18.8-1-Debian <<>> @1.1.1.1 -t NS zonetransfer.me
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51377
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; EDE: 18 (Prohibited)
;; QUESTION SECTION:
;zonetransfer.me.               IN      NS

;; ANSWER SECTION:
zonetransfer.me.        7181    IN      NS      nsztm1.digi.ninja.
zonetransfer.me.        7181    IN      NS      nsztm2.digi.ninja.

;; Query time: 15 msec
;; SERVER: 1.1.1.1#53(1.1.1.1) (UDP)
;; WHEN: Wed Feb 07 16:32:41 EST 2024
;; MSG SIZE  rcvd: 102
```

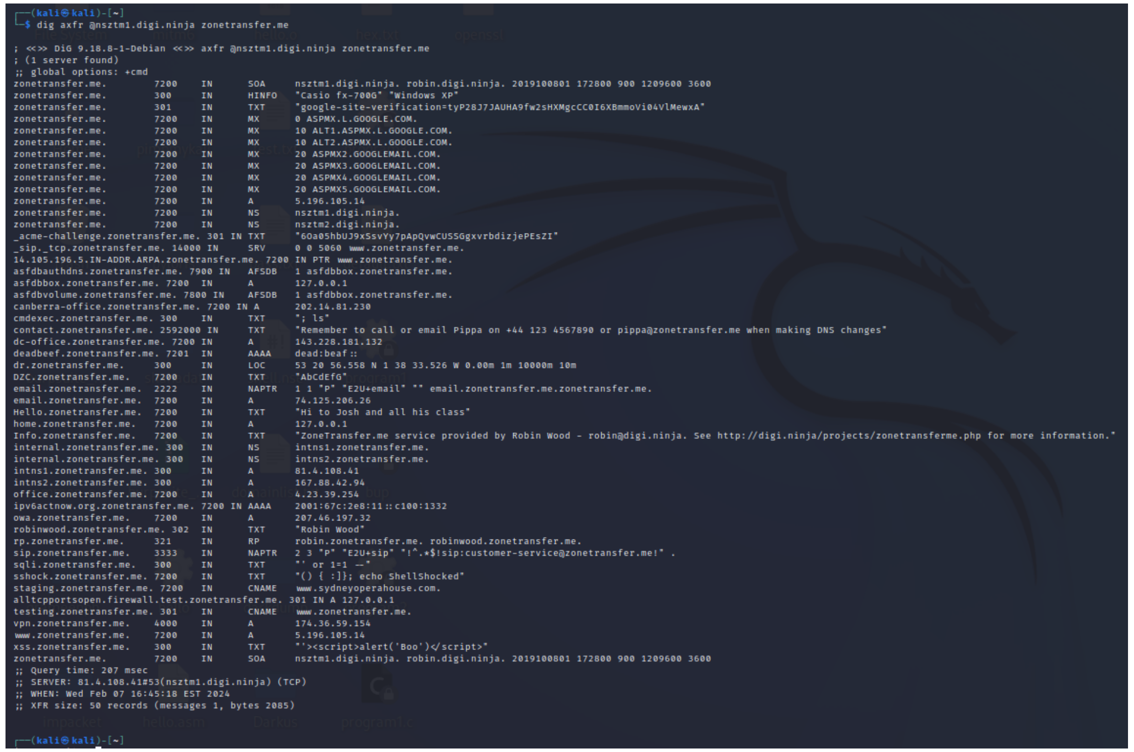
- To test the zone transfer, use the **$dig axfr @dnsserver zonetransfer.me**



```
┌──(kali㉿kali)-[~]
└─$ dig @1.1.1.1 -t NS +short zonetransfer.me
nsztm1.digi.ninja.
nsztm2.digi.ninja.

┌──(kali㉿kali)-[~]
└─$ 
```

Verifying the zone transfer configuration

- We will use **dig** command in this lab session
- Select the domain name for which we want to transfer the zone information. Our case it is **zonetransfer.me.** We resolve the name using DNS server 1.1.1.1



```
┌──(kali㉿kali)-[~]
└─$ dig axfr @nsztm1.digi.ninja zonetransfer.me

; <<>> DiG 9.18.8-1-Debian <<>> axfr @nsztm1.digi.ninja zonetransfer.me
; (1 server found)
;; global options: +cmd
zonetransfer.me.        7200    IN      SOA     nsztm1.digi.ninja. robin.digi.ninja. 2019100801 172800 900 1209600 3600
zonetransfer.me.        300     IN      HINFO   "Casio fx-700G" "Windows XP"
zonetransfer.me.        301     IN      TXT     "google-site-verification=tyP28J7JAUHA9fw2sHXMgcCC0I6XBmmoVi04VlMewxA"
zonetransfer.me.        7200    IN      MX      0 ASPMX.L.GOOGLE.COM.
zonetransfer.me.        7200    IN      MX      10 ALT1.ASPMX.L.GOOGLE.COM.
zonetransfer.me.        7200    IN      MX      10 ALT2.ASPMX.L.GOOGLE.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX2.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX3.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX4.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX5.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      A       5.196.105.14
zonetransfer.me.        7200    IN      NS      nsztm1.digi.ninja.
zonetransfer.me.        7200    IN      NS      nsztm2.digi.ninja.
_acme-challenge.zonetransfer.me. 301 IN TXT     "6Oa05hbUJ9xSsvYy7pApQvwCUSSGgxvrbdizjePEsZI"
_sip._tcp.zonetransfer.me. 14000 IN     SRV     0 0 5060 www.zonetransfer.me.
14.105.196.5.IN-ADDR.ARPA.zonetransfer.me. 7200 IN PTR www.zonetransfer.me.
asfdbauthdns.zonetransfer.me. 7900 IN   AFSDB   1 asfdbbox.zonetransfer.me.
asfdbbox.zonetransfer.me. 7200  IN      A       127.0.0.1
asfdbvolume.zonetransfer.me. 7800 IN    AFSDB   1 asfdbbox.zonetransfer.me.
canberra-office.zonetransfer.me. 7200 IN A      202.14.81.230
cmdexec.zonetransfer.me. 300    IN      TXT     "; ls"
contact.zonetransfer.me. 2592000 IN     TXT     "Remember to call or email Pippa on +44 123 4567890 or pippa@zonetransfer.me when making DNS changes"
dc-office.zonetransfer.me. 7200 IN      A       143.228.181.132
deadbeef.zonetransfer.me. 7201  IN      AAAA    dead:beaf::
dr.zonetransfer.me.     300     IN      LOC     53 20 56.558 N 1 38 33.526 W 0.00m 1m 10000m 10m
DZC.zonetransfer.me.    7200    IN      TXT     "AbCdEfG"
email.zonetransfer.me.  2222    IN      NAPTR   1 1 "P" "E2U+email" "" email.zonetransfer.me.zonetransfer.me.
email.zonetransfer.me.  7200    IN      A       74.125.206.26
Hello.zonetransfer.me.  7200    IN      TXT     "Hi to Josh and all his class"
home.zonetransfer.me.   7200    IN      A       127.0.0.1
Info.zonetransfer.me.   7200    IN      TXT     "ZoneTransfer.me service provided by Robin Wood - robin@digi.ninja. See http://digi.ninja/projects/zonetransferme.php for more information."
internal.zonetransfer.me. 300   IN      NS      intns1.zonetransfer.me.
internal.zonetransfer.me. 300   IN      NS      intns2.zonetransfer.me.
intns1.zonetransfer.me. 300     IN      A       81.4.108.41
intns2.zonetransfer.me. 300     IN      A       167.88.42.94
office.zonetransfer.me. 7200    IN      A       4.23.39.254
ipv6actnow.org.zonetransfer.me. 7200 IN AAAA    2001:67c:2e8:11::c100:1332
owa.zonetransfer.me.    7200    IN      A       207.46.197.32
robinwood.zonetransfer.me. 302  IN      TXT     "Robin Wood"
rp.zonetransfer.me.     321     IN      RP      robin.zonetransfer.me. robinwood.zonetransfer.me.
sip.zonetransfer.me.    3333    IN      NAPTR   2 3 "P" "E2U+sip" "!^.*$!sip:customer-service@zonetransfer.me!" .
sqli.zonetransfer.me.   300     IN      TXT     "' or 1=1 --"
sshock.zonetransfer.me. 7200    IN      TXT     "() { :]}; echo ShellShocked"
staging.zonetransfer.me. 7200   IN      CNAME   www.sydneyoperahouse.com.
alltcpportsopen.firewall.test.zonetransfer.me. 301 IN A 127.0.0.1
testing.zonetransfer.me. 301    IN      CNAME   www.zonetransfer.me.
vpn.zonetransfer.me.    4000    IN      A       174.36.59.154
www.zonetransfer.me.    7200    IN      A       5.196.105.14
xss.zonetransfer.me.    300     IN      TXT     "'><script>alert('Boo')</script>"
zonetransfer.me.        7200    IN      SOA     nsztm1.digi.ninja. robin.digi.ninja. 2019100801 172800 900 1209600 3600
;; Query time: 207 msec
;; SERVER: 81.4.108.41#53(nsztm1.digi.ninja) (TCP)
;; WHEN: Wed Feb 07 16:45:18 EST 2024
;; XFR size: 50 records (messages 1, bytes 2085)

┌──(kali㉿kali)-[~]
```

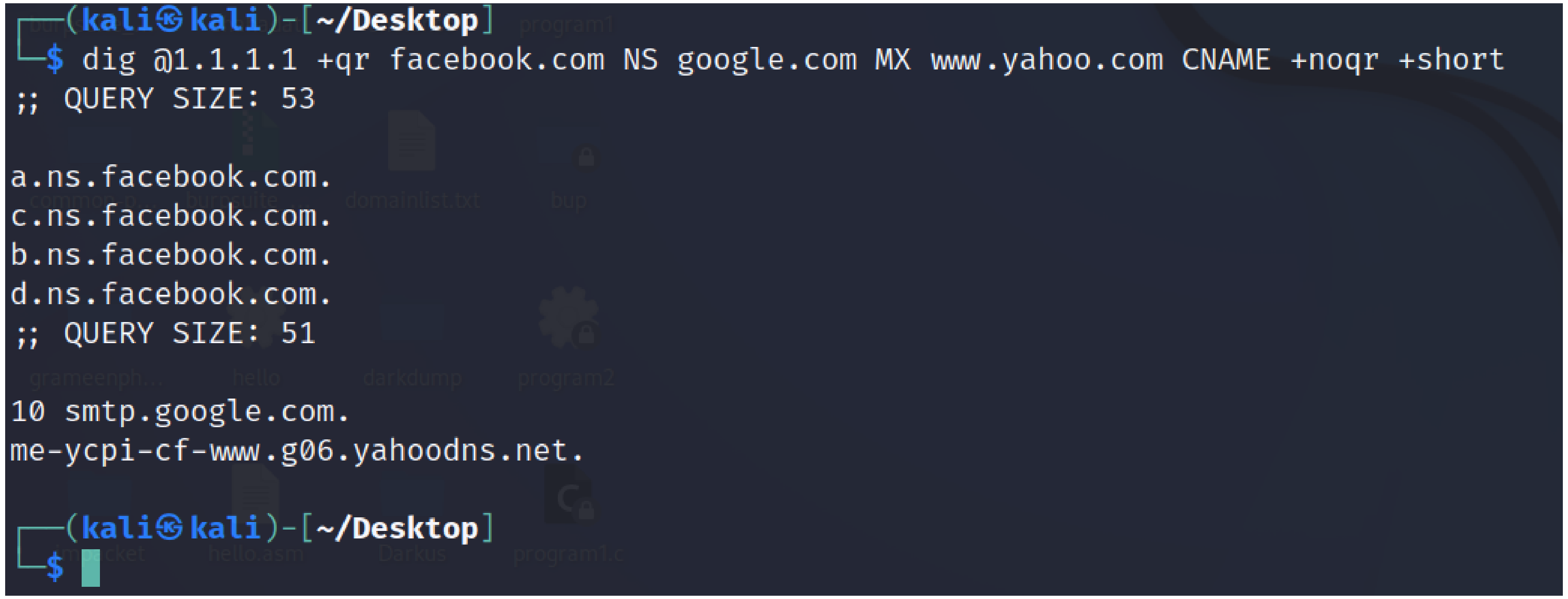
Test the zone transfer configuration for domain zonetransfer.me

- To test the zone transfer, use the
  $dig axfr @nsztm1.digi.ninja zonetransfer.me

Multiple query in a single command

The dig command take multiple query in a single command line.

Try below command
dig @1.1.1.1 +qr facebook.com NS google.com MX www.yahoo.com CNAME +noqr +short

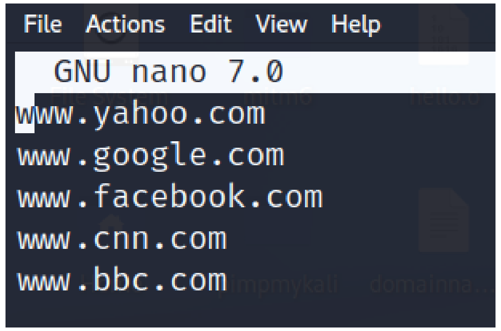


```
┌──(kali㉿kali)-[~/Desktop]
└─$ dig @1.1.1.1 +qr facebook.com NS google.com MX www.yahoo.com CNAME +noqr +short
;; QUERY SIZE: 53

a.ns.facebook.com.
c.ns.facebook.com.
b.ns.facebook.com.
d.ns.facebook.com.
;; QUERY SIZE: 51

10 smtp.google.com.
me-ycpi-cf-www.g06.yahoodns.net.

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

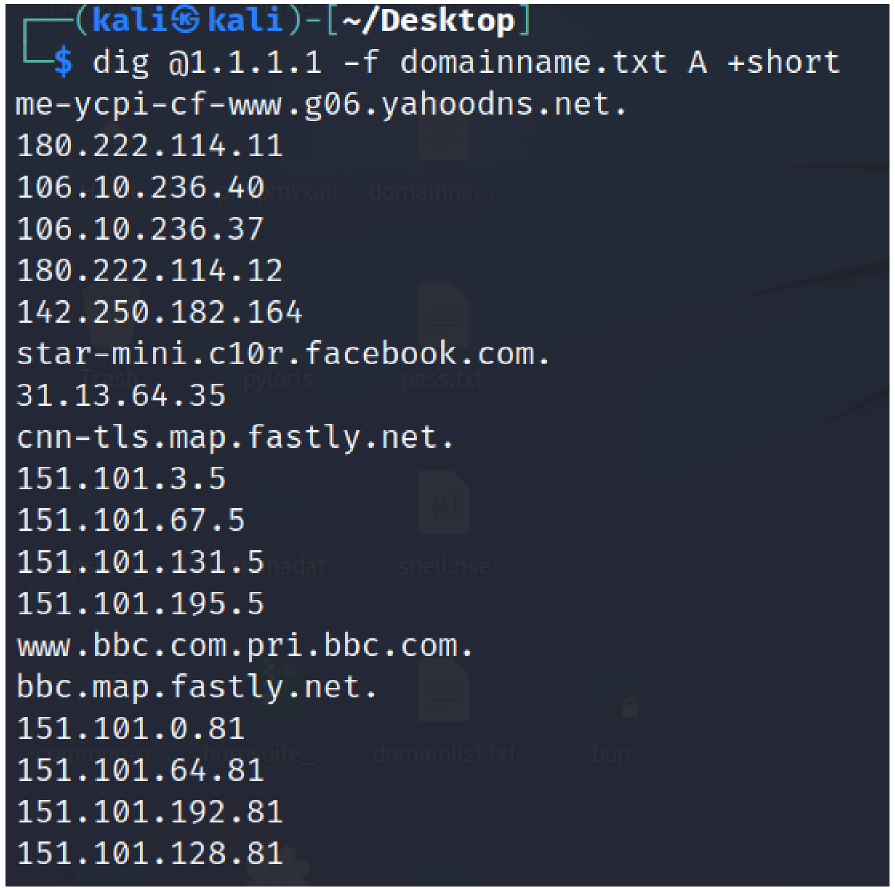
Batch mode query using dig

To query multiple domains at once by providing a list of domain names in a text file and then using the **-f** option to specify the file containing the list.

If a large number of hosts are to be determined by a lookup, then the host names can be inserted into one file - one name per line. With the -f option, the query can be performed one after the other.



```
  GNU nano 7.0
www.yahoo.com
www.google.com
www.facebook.com
www.cnn.com
www.bbc.com
```



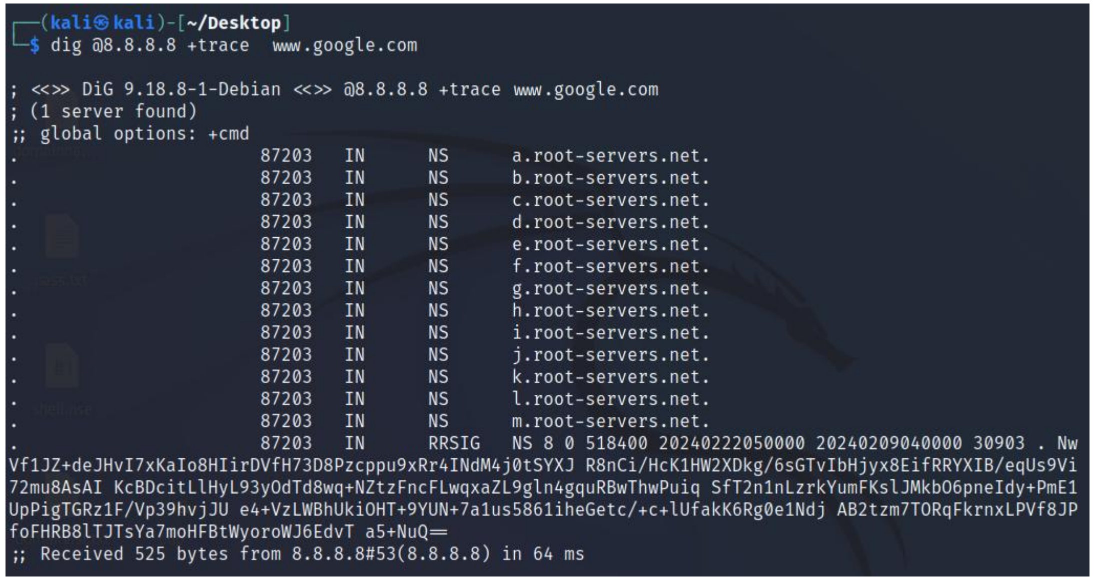
```
┌──(kali㉿kali)-[~/Desktop]
└─$ dig @1.1.1.1 -f domainname.txt A +short
me-ycpi-cf-www.g06.yahoodns.net.
180.222.114.11
106.10.236.40
106.10.236.37
180.222.114.12
142.250.182.164
star-mini.c10r.facebook.com.
31.13.64.35
cnn-tls.map.fastly.net.
151.101.3.5
151.101.67.5
151.101.131.5
151.101.195.5
www.bbc.com.pri.bbc.com.
bbc.map.fastly.net.
151.101.0.81
151.101.64.81
151.101.192.81
151.101.128.81
```

Tracking DNS communication

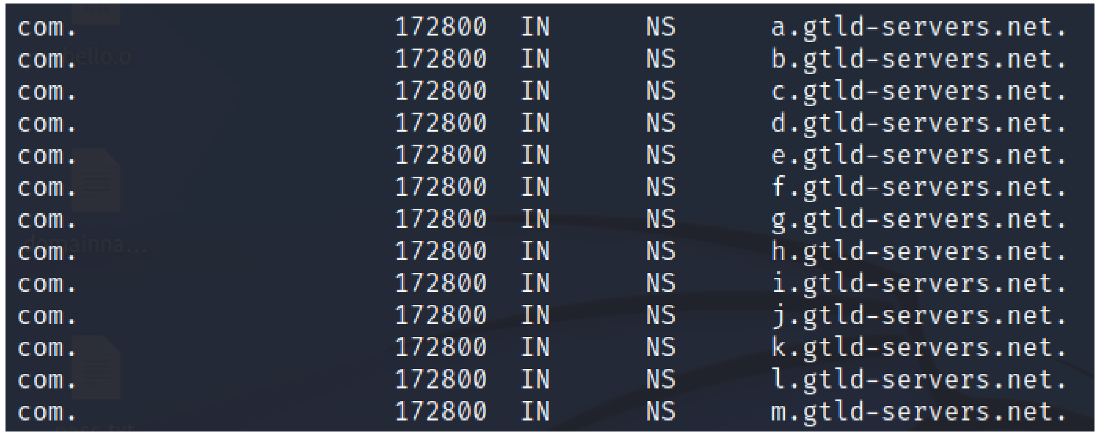
Anyone can use **+trace** option to trace the entire DNS server path from the root servers down to the authoritative name servers for the queried domain. This can help diagnose DNS resolution issues and understand the DNS hierarchy.

Example:

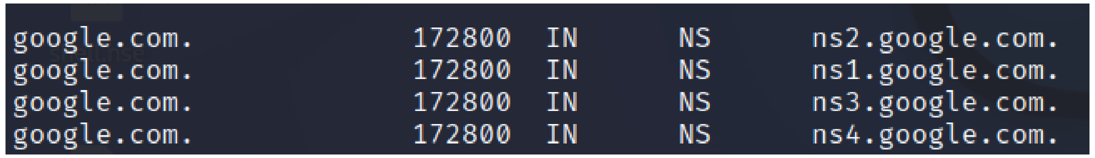
dig @1.1.1.1 google.com +trace



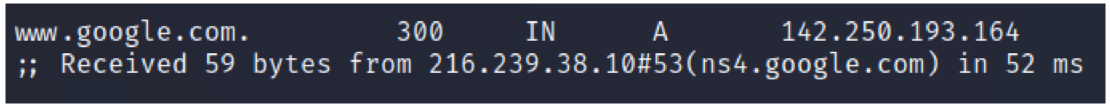
```
┌──(kali㉿kali)-[~/Desktop]
└─$ dig @8.8.8.8 +trace  www.google.com

; <<>> DiG 9.18.8-1-Debian <<>> @8.8.8.8 +trace www.google.com
; (1 server found)
;; global options: +cmd
.                       87203   IN      NS      a.root-servers.net.
.                       87203   IN      NS      b.root-servers.net.
.                       87203   IN      NS      c.root-servers.net.
.                       87203   IN      NS      d.root-servers.net.
.                       87203   IN      NS      e.root-servers.net.
.                       87203   IN      NS      f.root-servers.net.
.                       87203   IN      NS      g.root-servers.net.
.                       87203   IN      NS      h.root-servers.net.
.                       87203   IN      NS      i.root-servers.net.
.                       87203   IN      NS      j.root-servers.net.
.                       87203   IN      NS      k.root-servers.net.
.                       87203   IN      NS      l.root-servers.net.
.                       87203   IN      NS      m.root-servers.net.
.                       87203   IN      RRSIG   NS 8 0 518400 20240222050000 20240209040000 30903 . Nw
Vf1JZ+deJHvI7xKaIo8HIirDVfH73D8Pzcppu9xRr4INdM4j0tSYXJ R8nCi/HcK1HW2XDkg/6sGTvIbHjyx8EifRRYXIB/eqUs9Vi
72mu8AsAI KcBDcitLlHyL93yOdTd8wq+NZtzFncFLwqxaZL9gln4gquRBwThwPuiq SfT2n1nLzrkYumFKslJMkbO6pneIdy+PmE1
UpPigTGRz1F/Vp39hvjJU e4+VzLWBhUkiOHT+9YUN+7a1us5861iheGetc/+c+lUfakK6Rg0e1Ndj AB2tzm7TORqFkrnxLPVf8JP
foFHRB8lTJTsYa7moHFBtWyoroWJ6EdvT a5+NuQ==
;; Received 525 bytes from 8.8.8.8#53(8.8.8.8) in 64 ms
```



```
com.                    172800  IN      NS      a.gtld-servers.net.
com.                    172800  IN      NS      b.gtld-servers.net.
com.                    172800  IN      NS      c.gtld-servers.net.
com.                    172800  IN      NS      d.gtld-servers.net.
com.                    172800  IN      NS      e.gtld-servers.net.
com.                    172800  IN      NS      f.gtld-servers.net.
com.                    172800  IN      NS      g.gtld-servers.net.
com.                    172800  IN      NS      h.gtld-servers.net.
com.                    172800  IN      NS      i.gtld-servers.net.
com.                    172800  IN      NS      j.gtld-servers.net.
com.                    172800  IN      NS      k.gtld-servers.net.
com.                    172800  IN      NS      l.gtld-servers.net.
com.                    172800  IN      NS      m.gtld-servers.net.
```



```
google.com.             172800  IN      NS      ns2.google.com.
google.com.             172800  IN      NS      ns1.google.com.
google.com.             172800  IN      NS      ns3.google.com.
google.com.             172800  IN      NS      ns4.google.com.
```

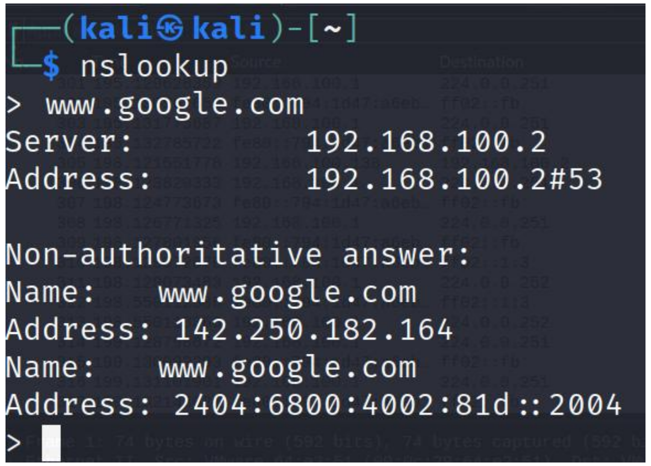


```
www.google.com.         300     IN      A       142.250.193.164
;; Received 59 bytes from 216.239.38.10#53(ns4.google.com) in 52 ms
```

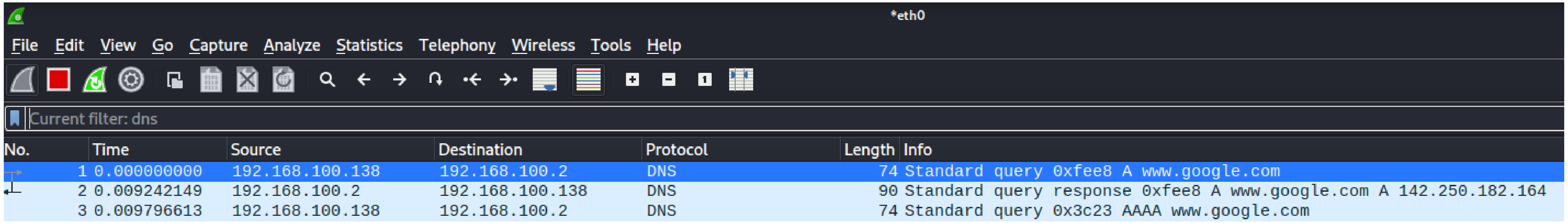
Analyzing the DNS traffic

Follow the below steps to identify the DNS query and response packet header

1. Run Wireshark
2. Generate a DNS query for www.google.com
3. Filter the DNS packets



```
┌──(kali㉿kali)-[~]
└─$ nslookup
> www.google.com
Server:         192.168.100.2
Address:        192.168.100.2#53

Non-authoritative answer:
Name:    www.google.com
Address: 142.250.182.164
Name:    www.google.com
Address: 2404:6800:4002:81d::2004
>
```



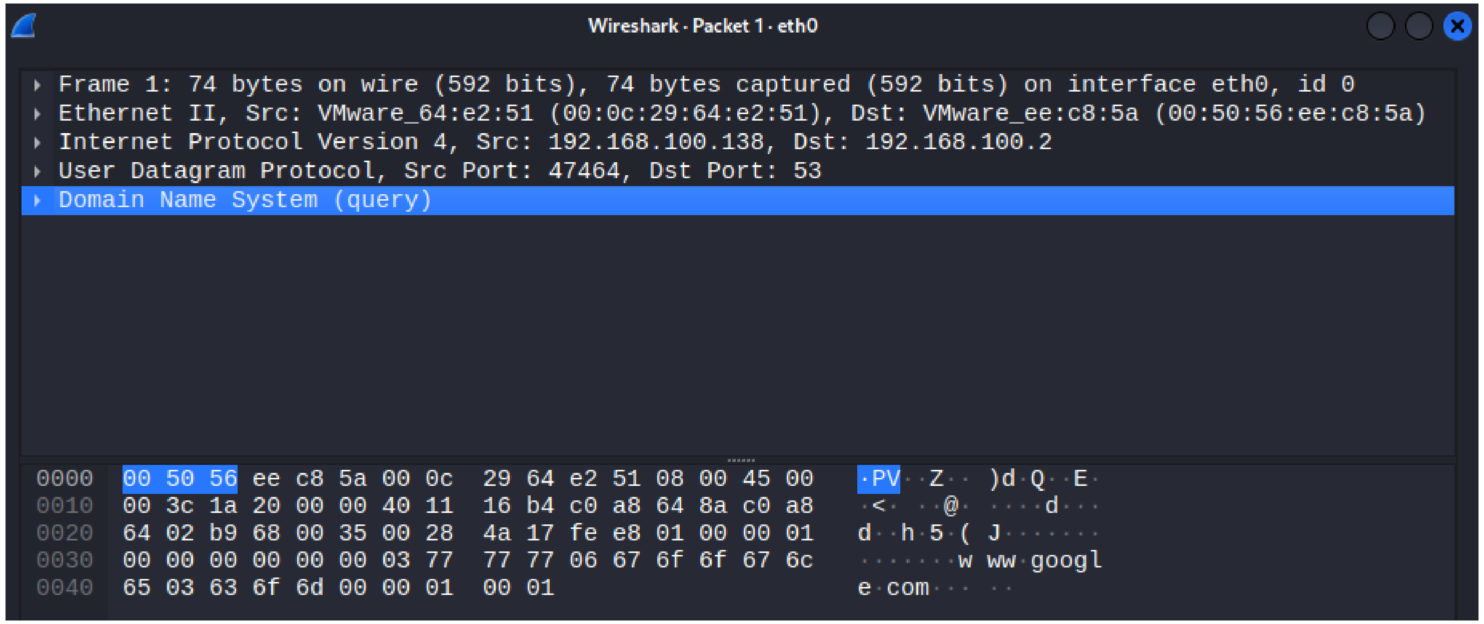
*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
Current filter: dns

No. Time         Source           Destination      Protocol  Length Info
  1 0.000000000  192.168.100.138  192.168.100.2    DNS           74 Standard query 0xfee8 A www.google.com
  2 0.009242149  192.168.100.2    192.168.100.138  DNS           90 Standard query response 0xfee8 A www.google.com A 142.250.182.164
  3 0.009796613  192.168.100.138  192.168.100.2    DNS           74 Standard query 0x3c23 AAAA www.google.com
```

Analyzing the DNS traffic

Analyze the Query packet

1. Double click the first packet



Wireshark · Packet 1 · eth0

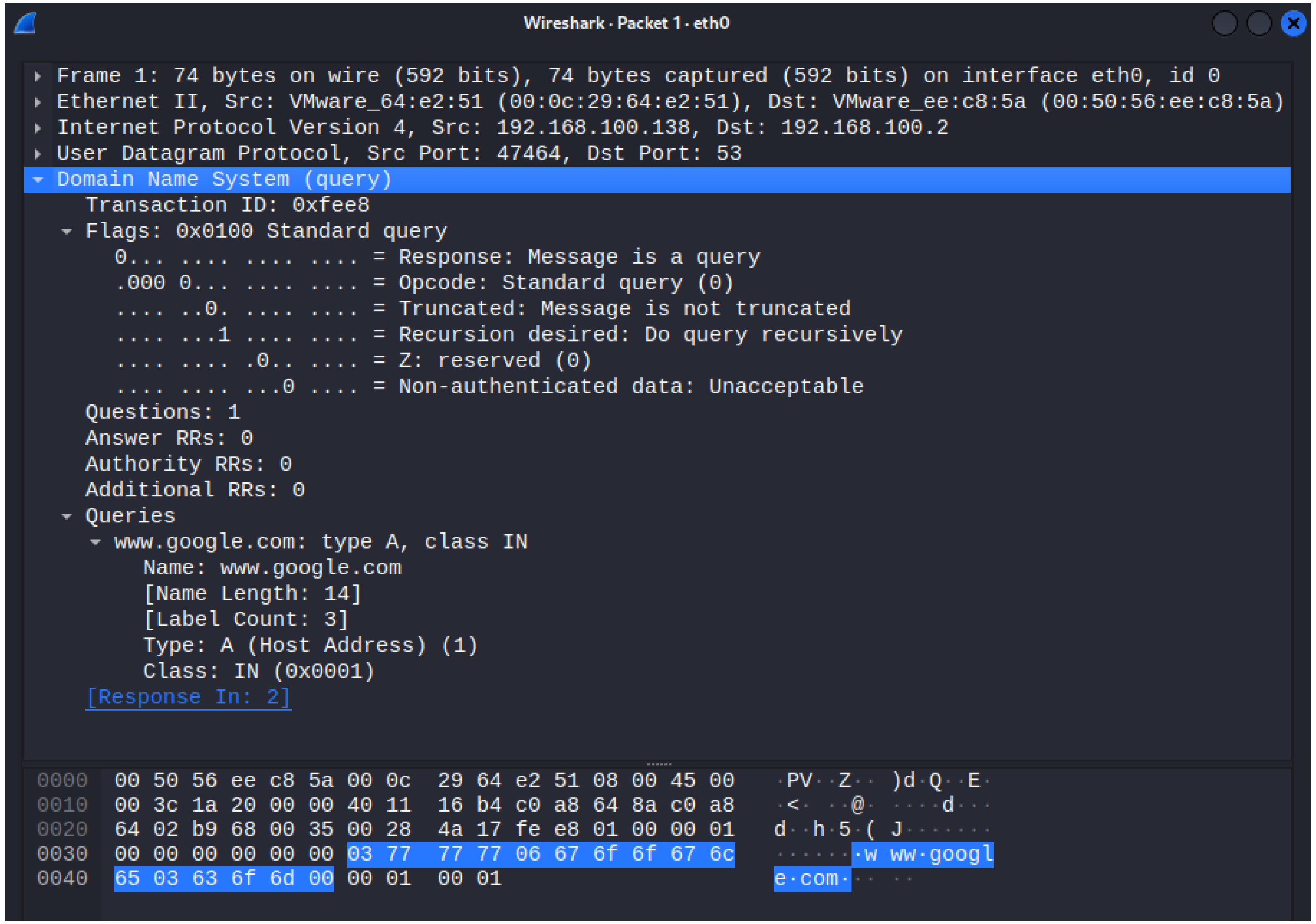
```
▸ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
▸ Ethernet II, Src: VMware_64:e2:51 (00:0c:29:64:e2:51), Dst: VMware_ee:c8:5a (00:50:56:ee:c8:5a)
▸ Internet Protocol Version 4, Src: 192.168.100.138, Dst: 192.168.100.2
▸ User Datagram Protocol, Src Port: 47464, Dst Port: 53
▸ Domain Name System (query)

0000  00 50 56 ee c8 5a 00 0c  29 64 e2 51 08 00 45 00   ·PV··Z·· )d·Q··E·
0010  00 3c 1a 20 00 00 40 11  16 b4 c0 a8 64 8a c0 a8   ·<· ··@· ····d···
0020  64 02 b9 68 00 35 00 28  4a 17 fe e8 01 00 00 01   d··h·5·( J·······
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   ·······w ww·googl
0040  65 03 63 6f 6d 00 00 01  00 01                     e·com··· ··
```

Analyzing the DNS traffic

Analyze the Query packet

1. Double click on the Domain Name System (query) packet

2. Expand the fields and get the explanation of each fields

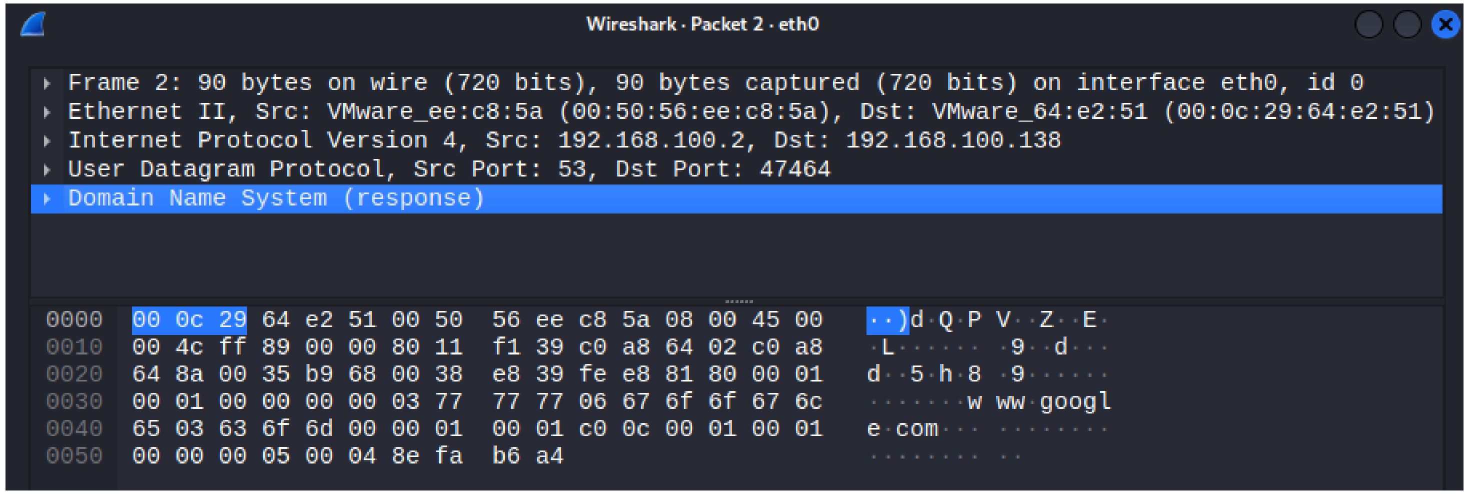


```
▸ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
▸ Ethernet II, Src: VMware_64:e2:51 (00:0c:29:64:e2:51), Dst: VMware_ee:c8:5a (00:50:56:ee:c8:5a)
▸ Internet Protocol Version 4, Src: 192.168.100.138, Dst: 192.168.100.2
▸ User Datagram Protocol, Src Port: 47464, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0xfee8
  ▾ Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ www.google.com: type A, class IN
        Name: www.google.com
        [Name Length: 14]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    [Response In: 2]

0000  00 50 56 ee c8 5a 00 0c  29 64 e2 51 08 00 45 00   ·PV··Z·· )d·Q··E·
0010  00 3c 1a 20 00 00 40 11  16 b4 c0 a8 64 8a c0 a8   ·<· ··@· ····d···
0020  64 02 b9 68 00 35 00 28  4a 17 fe e8 01 00 00 01   d··h·5·( J·······
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   ·······w ww·googl
0040  65 03 63 6f 6d 00 00 01  00 01                     e·com··· ··
```

Analyzing the DNS traffic

Analyze the Response Packet

1. Double click the second packet



Wireshark · Packet 2 · eth0

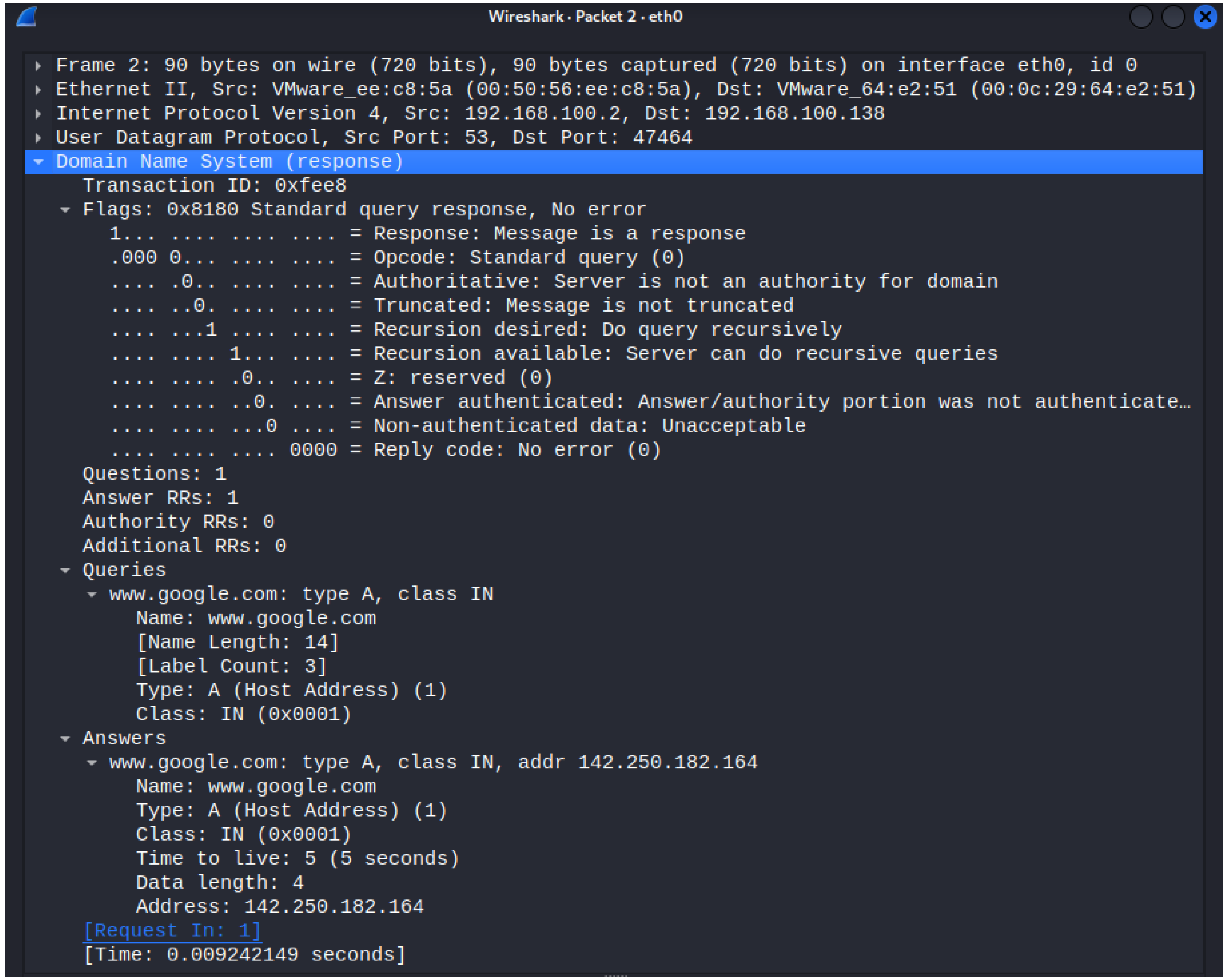
```
▸ Frame 2: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
▸ Ethernet II, Src: VMware_ee:c8:5a (00:50:56:ee:c8:5a), Dst: VMware_64:e2:51 (00:0c:29:64:e2:51)
▸ Internet Protocol Version 4, Src: 192.168.100.2, Dst: 192.168.100.138
▸ User Datagram Protocol, Src Port: 53, Dst Port: 47464
▸ Domain Name System (response)

0000  00 0c 29 64 e2 51 00 50  56 ee c8 5a 08 00 45 00   ··)d·Q·P V··Z··E·
0010  00 4c ff 89 00 00 80 11  f1 39 c0 a8 64 02 c0 a8   ·L······ ·9··d···
0020  64 8a 00 35 b9 68 00 38  e8 39 fe e8 81 80 00 01   d··5·h·8 ·9······
0030  00 01 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   ·······w ww·googl
0040  65 03 63 6f 6d 00 00 01  00 01 c0 0c 00 01 00 01   e·com··· ········
0050  00 00 00 05 00 04 8e fa  b6 a4                     ········ ··
```

Analyzing the DNS traffic

Analyze the Response Packet

1. Double click on the Domain Name System (Response) packet

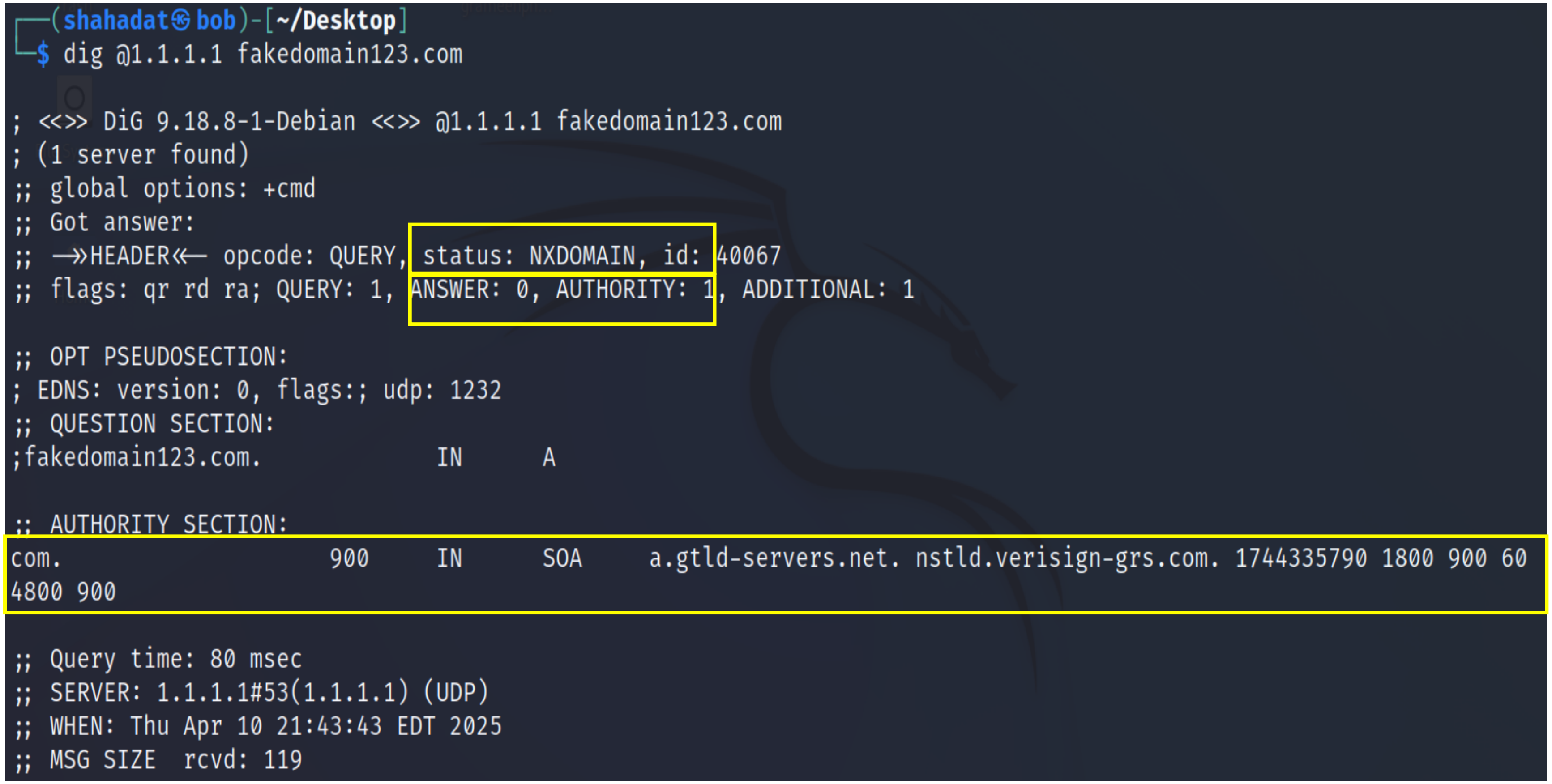
2. Expand the fields and get the explanation of each fields



Wireshark · Packet 2 · eth0

```
▸ Frame 2: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
▸ Ethernet II, Src: VMware_ee:c8:5a (00:50:56:ee:c8:5a), Dst: VMware_64:e2:51 (00:0c:29:64:e2:51)
▸ Internet Protocol Version 4, Src: 192.168.100.2, Dst: 192.168.100.138
▸ User Datagram Protocol, Src Port: 53, Dst Port: 47464
▾ Domain Name System (response)
    Transaction ID: 0xfee8
  ▾ Flags: 0x8180 Standard query response, No error
      1... .... .... .... = Response: Message is a response
      .000 0... .... .... = Opcode: Standard query (0)
      .... .0.. .... .... = Authoritative: Server is not an authority for domain
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... 1... .... = Recursion available: Server can do recursive queries
      .... .... .0.. .... = Z: reserved (0)
      .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticate…
      .... .... ...0 .... = Non-authenticated data: Unacceptable
      .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ www.google.com: type A, class IN
        Name: www.google.com
        [Name Length: 14]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
  ▾ Answers
    ▾ www.google.com: type A, class IN, addr 142.250.182.164
        Name: www.google.com
        Type: A (Host Address) (1)
        Class: IN (0x0001)
        Time to live: 5 (5 seconds)
        Data length: 4
        Address: 142.250.182.164
    [Request In: 1]
    [Time: 0.009242149 seconds]
```

Query to a Non-existence domain



```
┌──(shahadat㉿bob)-[~/Desktop]
└─$ dig @1.1.1.1 fakedomain123.com

; <<>> DiG 9.18.8-1-Debian <<>> @1.1.1.1 fakedomain123.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 40067
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;fakedomain123.com.             IN      A

;; AUTHORITY SECTION:
com.                    900     IN      SOA     a.gtld-servers.net. nstld.verisign-grs.com. 1744335790 1800 900 60
4800 900

;; Query time: 80 msec
;; SERVER: 1.1.1.1#53(1.1.1.1) (UDP)
;; WHEN: Thu Apr 10 21:43:43 EDT 2025
;; MSG SIZE  rcvd: 119
```

Performing Recursive and Iterative Query using dig

The dig command can be used to perform both recursive and iterative DNS queries.

Recursive Query:

A recursive query is the default behavior of dig. When a recursive query is made, the local DNS server (specified or implied) is asked to resolve the entire query. It will then perform the necessary steps, including contacting other DNS servers if needed, to provide the answer.

To make a recursive query using dig, simply specify the domain name:

`dig example.com` **OR**

To explicitly request recursion, you can use the +recurse option, though this is the default and often unnecessary:

```
dig +recurse example.com
```

Performing Recursive and Iterative Query using dig

Iterative Query:

An iterative query involves dig tracing the DNS resolution path by querying each DNS server in the chain, starting from the root servers, until the authoritative server for the domain is found. This provides a detailed view of the entire resolution process.

To make an iterative query using dig, use the **+trace** option:

```
dig +trace example.com
```

When using +trace, dig will automatically disable recursion, as it is performing the iterative steps itself. The output will show the responses from each server in the delegation chain, starting with the root servers and progressing through TLD servers to the authoritative name servers.

End of the Lab