REPORT

DNS Analysis with dig

v1.0.0

Author:

Eldon Gabriel

July 3, 2025

TABLE OF CONTENTS

REVISION HISTORY

Version	Date	Author	Description of Changes
v1.0.0	07/3/2025	Eldon G.	Initial draft.

SECTION 1.0: DNS Reconnaissance & Resolver Analysis

1.1 Project Description

This task involves using the `dig` tool on a Unix-based system to explore and query the Domain Name System (DNS). The goal is to learn how DNS works behind the scenes, including how domain names are translated into IP addresses. It also involves identifying which servers hold authority over a domain.

My job is to perform technical checks using command-line tools to gather DNS information. This helps improve organizational security skills by strengthening DNS enumeration. It is a valuable step in both attacking and defending network systems.

Source Acknowledgment:

Portions of this report are based on exercises from *Computer Networking: Principles, Protocols, and Practice* by Olivier Bonaventure (Apple Books). Content is used for educational and non-commercial purposes under fair use.

1.2 Resolver Identification & Basic Queries

Objective

This task helps identify which DNS server the system is using to answer name resolution requests. Knowing this is important for understanding how DNS traffic moves and where it could be monitored or tampered with.

Tools Used

- `dig` command in the terminal (Kali Linux)

Command Run

```bash
dig
```

What I Found

The output showed this line:

```yaml
;; SERVER: 192.168.64.1#53(192.168.64.1) (UDP)
```

This tells me that the system is using `192.168.64.1` as its DNS resolver. That IP is a private address, so it's likely coming from the local network, like a NAT adapter or a virtual machine bridge. It’s not a public DNS server like Google or Cloudflare.

Why It Matters

Using a local resolver can help speed up lookups, but it also means DNS queries might be handled or filtered by the host system. This is something attackers or defenders could use to their advantage.

Takeaway

Know which DNS server the system talks to. It helps in spotting misconfigurations, traffic leaks, or signs of spoofing or interception.

1.3 FQDN IP Resolution

Objective

This task checks how a Fully Qualified Domain Name (FQDN) resolves to an IP address. It shows if the name goes through any redirects (called CNAMEs) before reaching the final address. This helps with analyzing phishing links, subdomain takeovers, and finding hidden assets.

Tools Used

- `dig`
- `dig +short`

Commands Run

```bash
dig inl.info.ucl.ac.be
dig +short inl.info.ucl.ac.be
```

What I Found

The regular `dig` command gave the full DNS chain:

```pgsql
inl.info.ucl.ac.be.          CNAME     www.info.ucl.ac.be.
www.info.ucl.ac.be.          CNAME     info.ucl.ac.be.
info.ucl.ac.be.              A         130.104.228.147
```

This shows two CNAME records (which are like DNS redirects) that finally lead to an IPv4 address using an A record.

The `+short` version returned a simpler view:

```pgsql
www.info.ucl.ac.be.
info.ucl.ac.be.
130.104.228.147
```

This version is easier to read and useful in scripts or for quick checks.

Takeaway

FQDNs don't always go straight to an IP address. They often go through one or more CNAMEs. Learning to trace that chain is useful for tracking how websites are set up, spotting suspicious redirections, or uncovering hidden infrastructure.

1.4 Identifying Authoritative Nameservers

Objective

This task finds out which DNS servers are in charge of the `.be` top-level domain (TLD). These servers hold the official records for any domain ending in `.be`. Knowing who manages a domain's DNS can help detect misconfigurations or domain misuse.

Tools Used

- `dig`

Commands Run

```bash
dig -t NS be.
```

What I Found

The results show the six authoritative nameservers for the `.be` domain:

```css
be. IN NS a.nsset.be.
be. IN NS b.nsset.be.
be. IN NS c.nsset.be.
be. IN NS d.nsset.be.
be. IN NS y.nsset.be.
be. IN NS z.nsset.be.
```

These are the DNS servers trusted to give the correct answers for any `.be` domain. They are usually managed by the registry responsible for that TLD.

Takeaway

Authoritative nameservers are the final source of truth for DNS records. If they're misconfigured or compromised, entire domains can break or be redirected. This is why identifying and validating them is important during DNS investigations or threat hunting.

1.5 Root DNS Infrastructure Check

Objective

This task checks if the system can reach the root DNS servers — the top-level servers in the DNS system. They help direct queries to the right TLD servers (like `.com`, `.org`, `.be`, etc.).

Tools Used

- `dig`

Commands Run

```bash
dig
```

What I Found

Running `dig` with no arguments triggers a query to the root zone (`.`). The result included 13 different root name servers, which is expected:

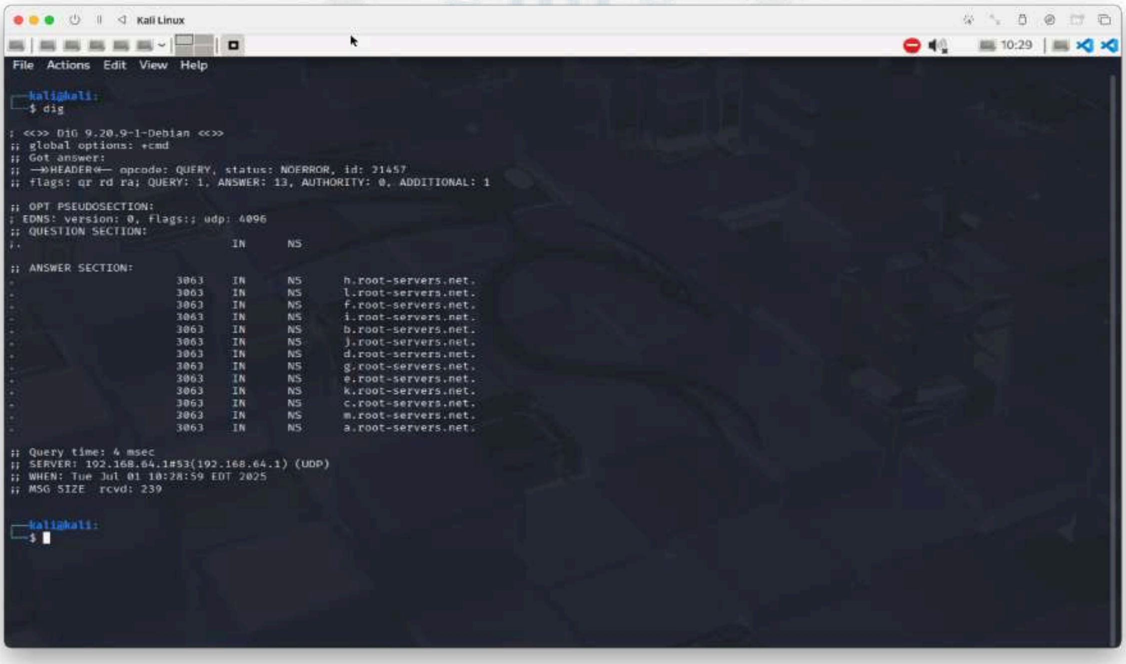


Figure 1: Default `dig` command output screenshot, July 2, 2025, *Kali Linux, dig 9.20.9*.

These root servers are spread globally and are key to how the DNS system works.

Takeaway

Access to all 13 root servers confirms that the system's DNS setup is working correctly. These servers are the starting point for all DNS lookups on the Internet.

1.6 Enumerating Root DNS Server IPs (IPv4 and IPv6)

Objective

This task finds the IP addresses of the 13 root DNS servers. These servers are the starting points for all DNS lookups. Knowing their IPs helps me understand how DNS works and troubleshoot problems.

Tools Used

- `dig`

Commands Run

```bash
# Get IPv4 addresses for all root servers
for letter in a b c d e f g h i j k l m; do
  dig +short ${letter}.root-servers.net A
done

# Get IPv6 addresses for all root servers
for letter in a b c d e f g h i j k l m; do
  dig +short ${letter}.root-servers.net AAAA
done
```

Summary of Output

IPv4:

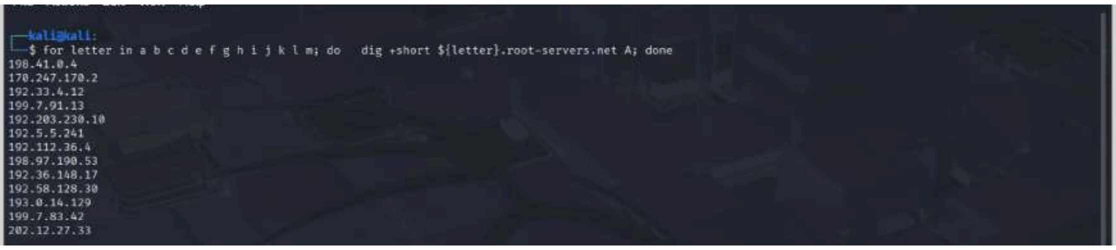


Figure 2: `dig` A loop showing root server IPv4s screenshot, July 2, 2025, *Kali Linux, dig 9.20.9*.

IPv6:

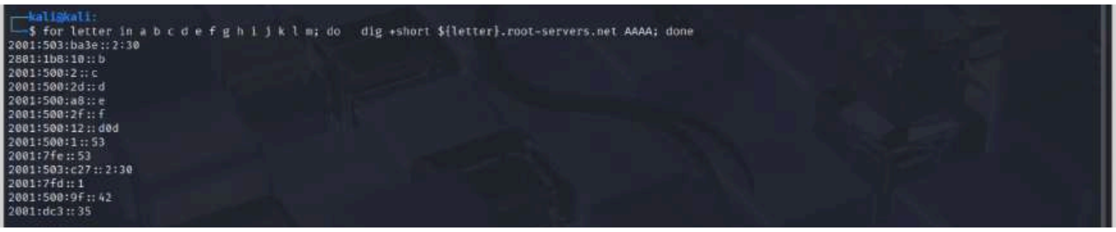


Figure 3: `dig` AAAA loop root servers screenshot, July 2, 2025, *Kali Linux, dig 9.20.9*.

Observations

- All 13 root servers return valid IPv4 addresses.
- Each server also returns a valid IPv6 address, showing support for modern IPv6 DNS queries.
- These IPs can be used to query the root servers directly over IPv4 or IPv6, depending on the network setup.

Extra Information

The official list of root server addresses is managed by IANA. You can get it anytime using:

```bash
curl -s http://www.internic.net/zones/named.root
```

This file helps DNS software start recursive lookups by providing “root hints.”

1.7 Discovering .org and root-servers.org NS

Objective

This task explores how DNS moves down the hierarchy. First, it finds the nameservers for the `.org` domain from a root server. Then it asks one of those `.org` nameservers for the nameservers of `root-servers.org`. This simulates how DNS looks up addresses step-by-step.

Tools Used

- `dig`

Commands Run

```bash
# Step 1: Get NS records for .org from a root server
dig @198.41.0.4 org. NS


# Step 2: Get NS records for root-servers.org from a .org nameserver
dig @199.19.56.1 root-servers.org. NS
```

Summary of Output

- Step 1 returned NS records for the .org domain from the root server.
- Example nameserver found: `a0.org.afilias-nst.info`
- Step 2 returned authoritative NS records for root-servers.org, including:
  - `a.icann-servers.net.`
  - `ns.maxgigapop.net.`
  - `ns-ext.isc.org.`
  - and others

- The response included IP addresses for `ns-ext.isc.org`:
    - **IPv4:** 149.20.2.126
    - **IPv6:** 2001:500:6b:2::126

Observations

- The root server correctly pointed to the `.org` zone servers.
- The `.org` nameserver gave the full list of nameservers for `root-servers.org`.
- The TTL for these NS records was 3600 seconds (1 hour), meaning DNS caches should refresh this data every hour.

1.8 Attempt to Resolve `root-servers.org`

Objective

After finding the authoritative nameservers for `www.root-servers.org`, this step tries to get the A record (IPv4 address) for `www.root-servers.org` by asking one of those servers. This is like the last step in how DNS looks up a website.

Tools Used

- `dig`

Commands Run

```bash
dig @199.7.83.42 www.root-servers.org A
```

Summary of Output

- The server at (`199.7.83.42`) replied with NS records for the `.org` domain, not the A record for `www.root-servers.org`.
- This means the server was not the right one to ask — it's not authoritative for `www.root-servers.org`.
- No A record was returned.

Observations

- The lookup failed because the wrong server was queried.
- To get the right answer, you need to query one of the correct authoritative servers, like `ns-ext.isc.org` at 149.20.2.126.

Next Step

Try this command to get the A record:

```bash
dig @149.20.2.126 www.root-servers.org A
```

This should return the IPv4 address and TTL, finishing the DNS lookup chain.

1.9 Final Resolution of www.root-servers.org

Objective

This step finished the full manual DNS lookup for `www.root-servers.org` by asking a known authoritative server. The goal was to get the A record (IPv4 address) and see the TTL (time the record is valid).

Tools Used

- `dig`

Commands Run

```bash
dig @149.20.2.126 www.root-servers.org A
```

Summary of Output

The server replied with:

```css
;; ANSWER SECTION:

www.root-servers.org.  3600  IN  A  193.0.11.23
```

- The A record shows the IP address is `193.0.11.23`
- TTL is 3600 seconds, which means the record can be cached for one hour.
- The response included the `aa` flag, meaning the server is authoritative for this domain.

Observations

- The recursion flag was ignored, as expected, because the server is authoritative and doesn't do recursion.
- The query took about 236 ms, showing a typical delay for direct DNS queries.
- The TTL confirms that recursive resolvers can keep this IP cached for an hour before needing to ask again.

1.10 DNS TTL Behavior for www.google.com

Objective

This task looks at how long DNS results are saved (cached) by the system using TTL (Time-To-Live). I tested it using www.google.com, which is a global service that changes fast and often.

Tools Used

- `dig`
- `sleep` (to wait between queries)

Commands Run

```bash
dig www.google.com A
sleep 5
dig www.google.com A
```

Summary of Output

First Result:

```css
www.google.org.   223    IN      A       142.250.199.36
```

After 5 seconds:

```css
www.google.org.   218    IN      A       142.250.199.36
```

- The IP stayed the same.
- The TTL dropped by 5 seconds, and the system cached the result.
- DNS server used: `192.168.64.1` (local resolver).

Observations

- Google uses short TTLs (starting around 300 seconds) so it can change traffic routes quickly.
- The result was saved in the cache and reused.
- This reduces the delay and saves system resources.

Why TTL Matters

- **Faster Browsing**: No need to look up names again right away
- **Less Load**: Fewer requests to DNS servers
- **Better Control**: Changes can take effect quickly when needed
- **Backup Plan**: Cached data can keep things running if DNS goes offline

1.11 Investigating MX Records and TTLs

Objective

This task checks how two domains: uclouvain.be and gmail.com—handle email delivery using **MX (Mail Exchange)** records. It also interprets the **TTL** (Time-To-Live) values to see how long these records are cached.

Tools Used

```bash
dig +ttlid uclovuvian.be MX
Dig +ttlid www.gmail.com MX
```

Summary of Output

uclouvain.be

```yaml
uclouvain.be.  4502  IN  MX  1
uclouvain-be.mail.protection.outlook.com.
```

- TTL: 4502 seconds (~1.25 hours)
- Only one MX record
- Preference value: **1** (highest priority)
- Points to Microsoft Outlook's mail system

gmail.com

```yaml
gmail.com.  2999  IN  MX  5 gmail-smtp-in.l.google.com.
gmail.com.  2999  IN  MX 10 alt1.gmail-smtp-in.l.google.com.
gmail.com.  2999  IN  MX 20 alt2.gmail-smtp-in.l.google.com.
gmail.com.  2999  IN  MX 30 alt3.gmail-smtp-in.l.google.com.
gmail.com.  2999  IN  MX 40 alt4.gmail-smtp-in.l.google.com.
```

- TTL: 2999 seconds (~50 minutes)

- Five MX records with different **preference values**
- Lower value = higher priority
- Starts with priority **5**, then uses backups if needed

Observations

- **UCLouvain** uses Microsoft 365 with a single, stable mail server.
- **Gmail** sends its email delivery across five servers for better uptime and faster response in different regions.
- TTL values reduce the number of DNS requests during email delivery. This improves performance and reliability.

1.12 Querying AAAA (IPv6) DNS Records

Objective

This task checks if websites support IPv6 by using `dig` to run AAAA record queries. The goal is to determine if the domains return valid IPv6 addresses, how the records are structured, and what their TTL (Time-To-Live) values are.

Tools/Queries Used

```bash
dig www.sixxs.net AAAA
dig www.google.com AAAA
dig ipv6.google.com AAAA
```

Summary of Output

www.sixxs.net

- CNAME: `www.sixxs.net` → `sixxs.net`
- AAAA Records:
  - `2001:7b8:3:1e::5`
  - `2a02:898:146::2`
  - `2a10:fc42:d::248`
- TTL: 473 seconds

www.google.com

- AAAA Record: `2404:6800:4001:811::20`
- TTL: 275 seconds

ipv6.google.com

- CNAME: `ipv6.google.com` → `ipv6.l.google.com`

- AAAA Record: `2404:6800:4001:80b::200e`

- TTLs: 429 (CNAME), 252 (AAAA)

Observations

- All domains returned valid IPv6 addresses.
- CNAME records were used to point to the actual servers.
- TTL values were short, especially for Google, which likely uses this for fast updates and load balancing.
- This confirms IPv6 support and shows how domains are mapped to IPs in modern DNS setups.

1.14 DNS Query ID Randomization

Objective

This task checks if the `dig` tool in Kali Linux uses random 16-bit IDs for DNS queries. These IDs help match replies to the correct request. If they are predictable, attackers can fake responses and poison DNS caches. Randomizing the ID makes spoofing much harder.

Method

Each DNS query includes a number called a "transaction ID." This ID should be different every time. To test this, I ran the same `dig` command several times with short delays in between to check if the IDs changed.

Tool Used:

- `dig` (DNS Lookup Tool)

Common Sequence:

```bash
dig -t MX gmail.com
sleep 1
dig -t MX gmail.com
sleep 1
dig -t MX gmail.com
```

Observed Transaction IDs:

- **First run:** id: `34211`
- **Second run**: id: `50852`
- **Third run:** id: `24599`
- **Fourth run:** id: `12290`
- **Fifth run:** id: `61626`
- **Sixth run:** id: `1310`

Conclusion

The `dig` tool on Kali Linux randomizes the DNS transaction ID in each query. This means it follows a key security best practice.

Security Importance

Randomizing DNS IDs protects systems from spoofed DNS replies. Attackers would have to guess the right ID to trick the system. This method is even stronger when combined with random source ports and other DNS hardening techniques (as outlined in RFC 5452).

Extra Notes

Even though the DNS record's TTL (Time-To-Live) stayed the same across responses (around 4400–4473 seconds), the query ID still changed. This shows that the random ID has nothing to do with how long DNS data is cached—it changes every time for security.

1.15 DNS Query ID Randomization

Objective

This task looks at how DNS resolvers (like BIND or Unbound) check if the DNS reply matches the original request. This is done using something called a "transaction ID." Matching this number is important to stop fake DNS replies from attackers.

Why Matching the ID Matters

DNS uses UDP, which doesn't have built-in checks to track who's talking to whom. That's why the transaction ID in the DNS reply must match the ID from the request. If not, someone could send a fake reply with the wrong IP addresses and trick users.

Tools Used

- `dig` (DNS query client)
- `bind` / `unbound` (DNS resolvers)

What Could Go Wrong

DNS software might handle the transaction ID in three ways:

- **Case A – Fixed ID**

  The ID stays the same (e.g., `id: 1234`). Attackers can guess it easily.
  Risk: *Very high.* Attackers can spoof replies and trick the system.

- **Case B – Incremental ID**

  The ID goes up in order (e.g., `1001, 1002, 1003`). Attackers can predict the next one.
  Risk: *High.* still vulnerable to attacks.

- **Case C – Randomized ID**

  The ID changes randomly each time, with 65,536 possibilities.
  Risk: Much lower. Attackers rarely guess correctly.

1.15.1 Security Implications

Checking the DNS transaction ID is a simple but vital security step. Without it, attackers can:

- Poison the DNS cache
- Redirect users to fake sites
- Launch man-in-the-middle attacks

Better Security Needs:

- Randomized transaction IDs
- Randomized source ports
- DNSSEC for strong data validation

These follow the security standards in RFC 5452.

1.16 Querying Root DNS Servers via UDP and TCP

Objective

This task tests whether DNS replies are faster over UDP or TCP. It uses the `dig` tool to ask a root DNS server for a list of name servers.

Tools Used

- `dig` (version 9.20.9-1-Debian)
- Root DNS server: `192.33.4.12` (C-root)

Commands Run

```bash
dig @192.33.4.12 . NS
dig +tcp @192.33.4.12 . NS
```

Issues & Fixes

- No problems occurred.
- UDP returned a normal response with no recursion (this is expected from root servers).
- TCP also worked and gave back the same result.

Both methods returned:

- **13 NS records** (name servers for the root zone)
- **27 additional records** (IPv4 and IPv6 addresses for those name servers)

1.16.1 Screenshots

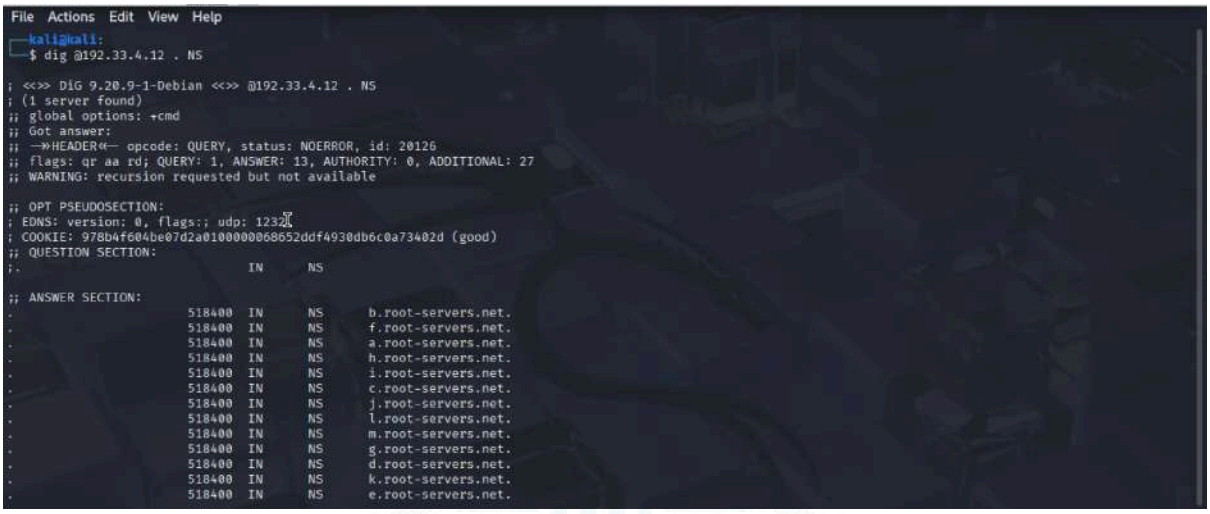


Figure 4: DNS query header and initial NS records. *Source: Kali Linux, dig 9.20.9*.

Figure 5: More NS records and IPv4 addresses. *Source: Kali Linux, dig 9.20.9*.

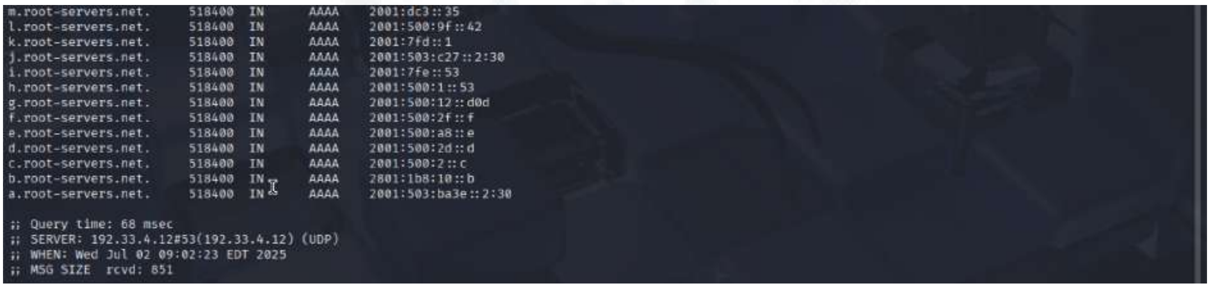
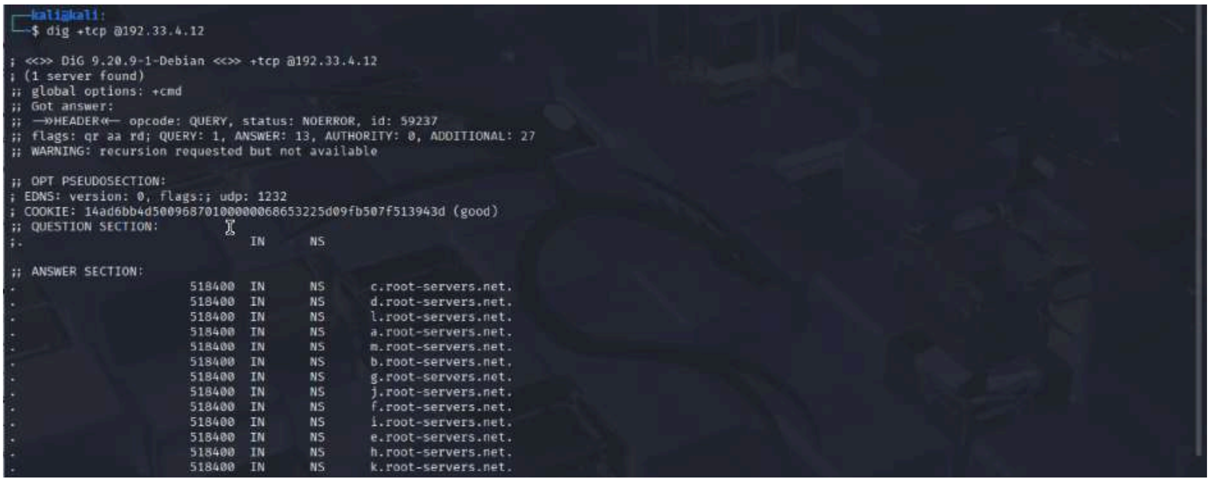


Figure 6: IPv6 addresses and query details. *Source: Kali Linux, dig 9.20.9*.

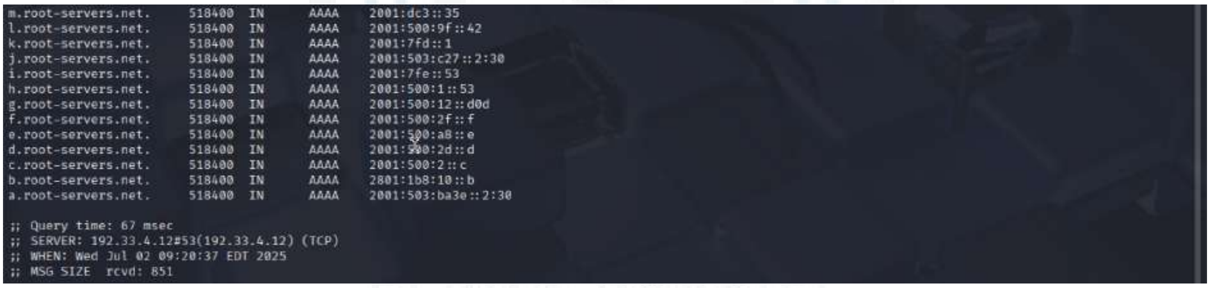


```
┌──kali@kali:
└─$ dig +tcp @192.33.4.12

; <<>> DiG 9.20.9-1-Debian <<>> +tcp @192.33.4.12
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59237
;; flags: qr aa rd; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 27
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 14ad6bb4d50096870100000068653225d09fb507f513943d (good)
;; QUESTION SECTION:
;.                              IN      NS

;; ANSWER SECTION:
.                       518400  IN      NS      c.root-servers.net.
.                       518400  IN      NS      d.root-servers.net.
.                       518400  IN      NS      l.root-servers.net.
.                       518400  IN      NS      a.root-servers.net.
.                       518400  IN      NS      m.root-servers.net.
.                       518400  IN      NS      b.root-servers.net.
.                       518400  IN      NS      g.root-servers.net.
.                       518400  IN      NS      j.root-servers.net.
.                       518400  IN      NS      f.root-servers.net.
.                       518400  IN      NS      i.root-servers.net.
.                       518400  IN      NS      e.root-servers.net.
.                       518400  IN      NS      h.root-servers.net.
.                       518400  IN      NS      k.root-servers.net.
```

Figure 7: DNS query header and first batch of NS records over TCP. *Source: Kali Linux, dig 9.20.9*.

```
;; ADDITIONAL SECTION:
m.root-servers.net.     518400  IN      A       202.12.27.33
l.root-servers.net.     518400  IN      A       199.7.83.42
k.root-servers.net.     518400  IN      A       193.0.14.129
j.root-servers.net.     518400  IN      A       192.58.128.30
i.root-servers.net.     518400  IN      A       192.36.148.17
h.root-servers.net.     518400  IN      A       198.97.190.53
g.root-servers.net.     518400  IN      A       192.112.36.4
f.root-servers.net.     518400  IN      A       192.5.5.241
e.root-servers.net.     518400  IN      A       192.203.230.10
d.root-servers.net.     518400  IN      A       199.7.91.13
c.root-servers.net.     518400  IN      A       192.33.4.12
b.root-servers.net.     518400  IN      A       170.247.170.2
a.root-servers.net.     518400  IN      A       198.41.0.4
```

Figure 8: Remaining NS records with IPv4 addresses over TCP. *Source: Kali Linux, dig 9.20.9*.



```
m.root-servers.net.     518400  IN      AAAA    2001:dc3::35
l.root-servers.net.     518400  IN      AAAA    2001:500:9f::42
k.root-servers.net.     518400  IN      AAAA    2001:7fd::1
j.root-servers.net.     518400  IN      AAAA    2001:503:c27::2:30
i.root-servers.net.     518400  IN      AAAA    2001:7fe::53
h.root-servers.net.     518400  IN      AAAA    2001:500:1::53
g.root-servers.net.     518400  IN      AAAA    2001:500:12::d0d
f.root-servers.net.     518400  IN      AAAA    2001:500:2f::f
e.root-servers.net.     518400  IN      AAAA    2001:500:a8::e
d.root-servers.net.     518400  IN      AAAA    2001:500:2d::d
c.root-servers.net.     518400  IN      AAAA    2001:500:2::c
b.root-servers.net.     518400  IN      AAAA    2801:1b8:10::b
a.root-servers.net.     518400  IN      AAAA    2001:503:ba3e::2:30

;; Query time: 67 msec
;; SERVER: 192.33.4.12#53(192.33.4.12) (TCP)
;; WHEN: Wed Jul 02 09:20:37 EDT 2025
;; MSG SIZE  rcvd: 851
```

Figure 9: IPv6 addresses and query metadata over TCP. *Source: Kali Linux, dig 9.20.9*.

1.16.2 Takeaways & Recommendations

UDP query time: ~68 ms
TCP query time: ~67 ms

Conclusion

Normally, UDP is faster than TCP because it doesn't need to set up a connection. But in this case, both queries took about the same time. The 1 millisecond difference is very small and could be caused by random network delays. So, there was no real speed difference in this test.

Security Context

TCP is better than UDP when:

- The DNS response is too big for UDP (over ~512 bytes)
- You're using **DNSSEC** for secure DNS
- The response is **truncated** and needs a reliable connection

Also, root DNS servers **don't support recursion**, so this test only checks direct queries, not full DNS lookups.

Recommendation

Use **UDP** for regular DNS queries—it's fast and has low overhead.
Use **TCP** if:

- The response is cut off
- You need stronger DNS security (like DNSSEC)
- You're troubleshooting or need a reliable connection

SECTION 2.0: CONCLUSION

2.1 Key Takeaways

This project used the `dig` command in Kali Linux to explore how DNS works. It covered how queries are built, how DNS IDs are randomized, and how different protocols (UDP vs. TCP) behave. Here's what was learned:

- **Random IDs:** Each DNS query had a different transaction ID. This helps stop attacks like DNS spoofing or cache poisoning.
- **Root Server Behavior:** Root DNS servers only give direct answers. They don't support recursion. This is expected and matches how the DNS system is designed.
- **UDP vs. TCP:** UDP is usually faster, but here both took about the same time (UDP: 68 ms, TCP: 67 ms). The 1 ms difference doesn't matter much and could just be a normal network delay.
- **Root Server Responses:** When asking a root server for data, it returned 13 NS records and 27 A/AAAA records. This shows how root servers help guide queries through the DNS system.
- **TCP Reliability:** TCP worked well when used for DNS. It's useful for big responses or secure DNS features like DNSSEC.

Overall, this section helped confirm how DNS protocols behave under different conditions. It showed how small changes in configuration can make DNS more secure and reliable.

2.2 Security Implications and Recommendations

Risks Found

- If DNS queries use predictable IDs or ports, attackers can fake DNS replies.
- Not using TCP fallback might cause missed or broken DNS responses.
- Accepting DNS replies with mismatched IDs can let attackers send fake data.

2.2.1 Recommendations

Technical Actions

- Use randomized transaction IDs and source ports (per RFC 5452).
- Turn on DNSSEC on recursive DNS servers to check that responses are real.
- Use TCP when UDP doesn't work, especially if the response is large or cut off.
- Watch DNS traffic for signs of tampering, like strange TTL values or fake domains

Procedural Actions

- Teach IT staff how to configure DNS securely.
- Add DNS hardening steps to the organization's standard setup process.
- Test DNS servers regularly to check for weak configurations.

Matches with Security Standards

- **NIST SP 800-81 Rev. 2:** Recommends using random IDs, TCP fallback, and DNSSEC.
- **ISO/IEC 27001 (A.12.4.1):** Encourages logging DNS activity to detect threats.
- **PCI DSS v4.0 (Req. 1.2.6):** Requires secure DNS setups in systems handling credit card data.

Compliance Relevance

Any group that works with sensitive data (like banks or hospitals) needs strong DNS security. Following these recommendations helps meet rules like **PCI-DSS**, **ISO 27001**, and **NIST CSF**. It also helps protect users from fake websites and DNS attacks.
