The Application Layer: Domain Name System (DNS)

COMPUTER NETWORKS A.A. 24/25

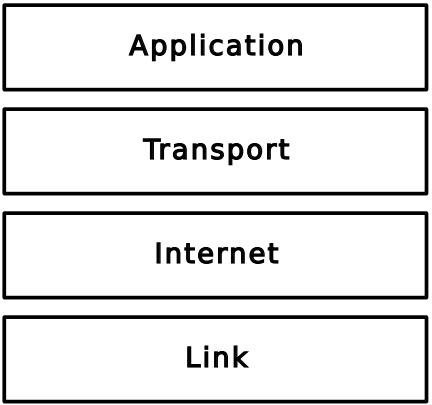
Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

Sect. 1 The Application Layer

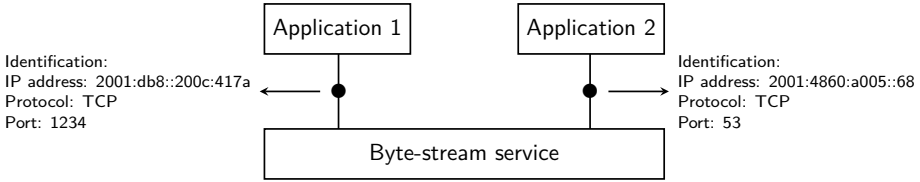
Application Layer Protocol

- The application layer implements protocols that offer specific services to the users
- These are for instance e-mail, web browsing, etc.
- The protocols must provide adequate procedures to perform the data-exchange that the application requires



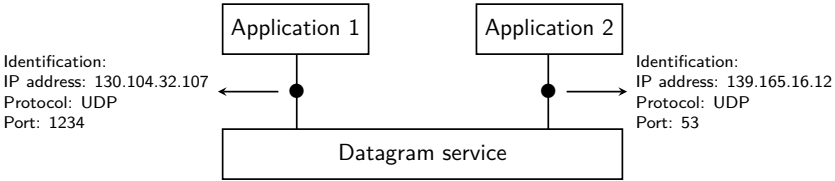
The Needed Service

- Application layer protocols rely on the transport layer, that as we know, can offer a connection-based or connection-less service
- There are several applications that can run on the same server, and each one is identified by a different *port number* offered by the transport layer.



TCP Vs UDP Transport

- The previous image represents the data that identify a TCP connection
- We will see that TCP is a transport-layer protocol, what is important to note at this stage is that when two applications communicate, their communication is uniquely identified by the two ports, and the two network addresses of the hosts.
- Exactly the same happens for a connection-less transport, that uses another protocol, UDP. In that case we don't use the term "connection".



IP Addresses

- Every NIC (Network Interface Card) that has access to the Internet needs a network layer address as we mentioned already
- These are normally referred to as IP addresses (where IP stands for Internet Protocol)
- The two figures use a different kind of IP address, that are both used in nowadays Internet, IPv4, and IPv6.
- We will talk about these protocols in future lessons, for the time being, all you need to know is the following

IP Addresses

- **An application runs on a server, that is identified by at least one IP address**
- **On the server, several applications run, each one identified by one port**
- **IP addresses are sequence of numbers made of 32 (v4) or 128 (v6) bits, i.e. 1.2.3.4**

Resolving Names

- However, when you access a webpage you are not typing an IP address and a port into your browser, you type a domain name
- There must be a way to map that name to an address and a port
- The fact that you are using a browser identifies the kind of application you are using, and this already decides the port. Browsers connect to a set of well known port numbers
- What is missing, is how you convert a name into an address
- So the first application we look at, it's the Domain Name System, that takes care of name conversion.

The Application Layer

↳1.1 DNS - Domain Name System

A Domain Name

- A domain name is a dotted separated string like `www.unive.it`
- A domain name has a hierarchical organization, from left (the lowest level) to right (the highest level)
- The highest level is called a Top Level Domain (TLD), and for many years there have been only a few allowed
- There is one organization, ICANN[1] that is responsible for assigning domains

[1] See `https://www.icann.org/`

TLD Delegation

- TLD are delegated by ICANN to other organizations, like national states
- These organization then further delegate to other organizations, that are called Domain Name Registrars[2].
- Registrars are normally large providers, or specialized companies.

[2] See `https://en.wikipedia.org/wiki/Domain_name_registrar`

TLDs

- There is one TLD for every UN recognized country (`.it`, `.de`, `.fr`...)
- There are other ones widely used (`.com`, `.org`, `.net`...)
- Recently, the TLD registration was opened up to any possible word, however, registering a second level domain costs less than 10€, while a new TLD costs at least 185,000$ [3]

[3] See `https://newgtlds.icann.org/en/applicants/global-support/faqs/faqs-en`

Domain Delegation

- When someone is given a certain domain, he is responsible for all the sub-domains
- So if the owner of `example.com` delegates `my.example.com` to some other identity, then this other identity can autonomously extend it, and eventually delegate again
- This makes it possible to scale the domain name tree without a central control
- A sub-tree of the whole domain tree is called a *zone*, and we talk about *zone delegation*: `www.example.com`, `main.example.com`, `ns.example.com` are all domains in the same zone.
- A server authoritative for a zone, can also be outside the zone itself.

How Do I Register a Name?

- Imagine someone want to set-up a new web page on the domain `example.com`
- He needs a server where he will place the web page (a web page is a file), and a public IP address, let's say he owns the IP `1.2.3.5`
- He needs to set-up a web server on the computer that uses `1.2.3.5`. This is an application that will serve the page when requested.
- Then he makes an account on some registrar, and if the domain is available, he pays the lease (domains are always leased for some time, they are not perpetual), and at that point he owns the domain.

How Do I Register a Name? (2)

- Then he has to configure a DNS server
- This is the server that will answer the name queries for the domain, providing the match between name and IP
- This requires another server, with a public IP, but normally the registrar can offer this service, so he just has to add some configuration on the registrar DNS server.
- So the DNS server of the registrar with a public IP, let's say `1.2.3.4` will know the association: `example.com` $\rightarrow$ `1.2.3.5`
- Finally, the registrar will tell to the TLD owner for `.com` that there is a new sub domain called `example`. This makes the registrar DNS server *authoritative* for that domain.

Accessing the Domain

- A web browser wants to browse `example.com`
- The browser does not know who is authoritative for it. The browser instead knows a list of so-called *root servers*.
- The root servers are 13 severs spread around the world whose addresses are hardcoded into browsers.
- Root servers provide to the browser the address of the DNS server that are authoritative for all TLDs.

Accessing the Domain

- The browser will contact the authoritative DNS server for `.com` and ask for the IP of `example.com`
- The authoritative domain will answer "*I don't know, but I know that 1.2.3.4 is authoritative for it*"
- Then the browser will query `1.2.3.4`, that will finally answer that `example.com` is at `1.2.3.5`

DNS Key Concepts

- The communication on the Internet happens only though IP addresses. Data packets never contain domain names
- A DNS server is a server that answers the question *what IP address corresponds to this domain?*
- DNS servers are organized in a tree, if a server does not know the answer, it may provide the IP address of another server
- The browser (or any other application) will first query a DNS server, and then, once it achieved the IP address of the server will open a transport layer connection to the right IP and port

Tests with dig

- Let's try: `dig` is a program to make DNS resolution
- the syntax is `dig @server target.domain`
- `198.41.0.4` is a root server
- lines starting with ";" are comments, they do not give information

```
1  $ dig @198.41.0.4 www.unive.it
2  ;; QUESTION SECTION:
3  ;www.unive.it.       IN   A
4  ;; AUTHORITY SECTION:
5  it.      172800   IN   NS   a.dns.it.
6  it.      172800   IN   NS   m.dns.it.
7  it.      172800   IN   NS   dns.nic.it.
8  it.      172800   IN   NS   d.dns.it.
9  it.      172800   IN   NS   nameserver.cnr.it.
10 it.      172800   IN   NS   r.dns.it.
```

There is no ANSWER section, so the server doesn't have an answer. Lines 4-10 says "I can tell you who to ask, ask `a.dns.it` that is authoritative for `.it`"

Tests with dig

- Let's ask to `a.dns.it`

```
$ dig @a.dns.it www.unive.it
;; QUESTION SECTION:
;www.unive.it.      IN   A

;; AUTHORITY SECTION:
unive.it.    3600   IN   NS   dns.cineca.it.
unive.it.    3600   IN   NS   ns1.garr.net.
unive.it.    3600   IN   NS   vega.unive.it.
unive.it.    3600   IN   NS   algol.unive.it.
```

Again no answer, but ask `dns.cineca.it` that is authoritative for `unive.it`. So the next resolution must be from an authoritative server.

Tests with dig

- Let's ask to `a.dns.it`

```
$ dig @dns.cineca.it www.unive.it
;; QUESTION SECTION:
;www.unive.it.      IN  A

;; ANSWER SECTION:
www.unive.it.   86400 IN  A 157.138.7.88

;; AUTHORITY SECTION:
unive.it.   86400 IN  NS  algol.unive.it.
unive.it.   86400 IN  NS  ns1.garr.net.
unive.it.   86400 IN  NS  vega.unive.it.
unive.it.   86400 IN  NS  dns.cineca.it.
```

We have an answer! `157.138.7.88` is www.unive.it. The `AUTHORITATIVE SECTION` will be clear in a minute.

A Very Inefficient Process

- The one I have depicted is a very inefficient process (albeit functional)
- Every time someone wants to browse a website a three-step query starts.

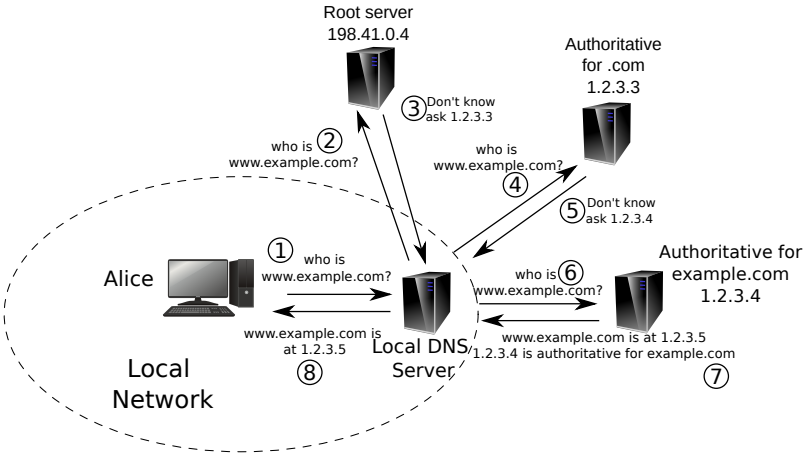
Most of the times the behaviour differs:

- Every Internet host must be configured with the IP of a DNS server. This is generally done automatically when you enter the network (more on this in the future)
- The DNS server is internal to the network
- That host will always query the internal DNS server, that will query other servers

- So let's see what happens when Alice enters a network and wants to browse `www.example.com`[4]

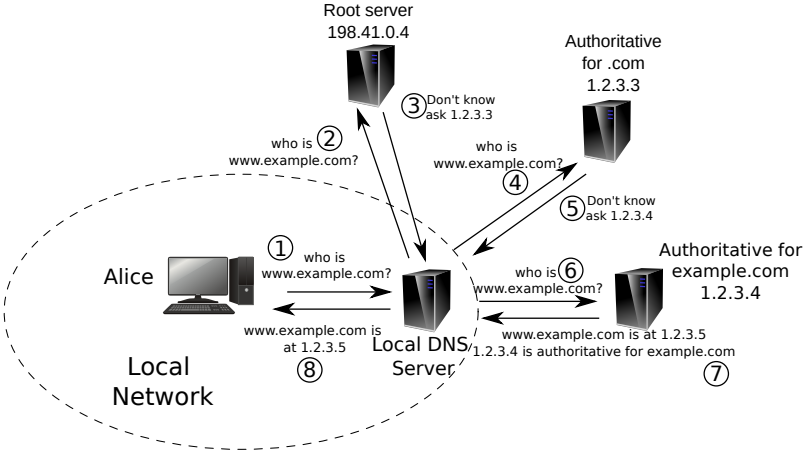
[4] Note that `www` is nothing more than a part of a domain. It does not have any special meaning, it is just a convention for web pages.

DNS query



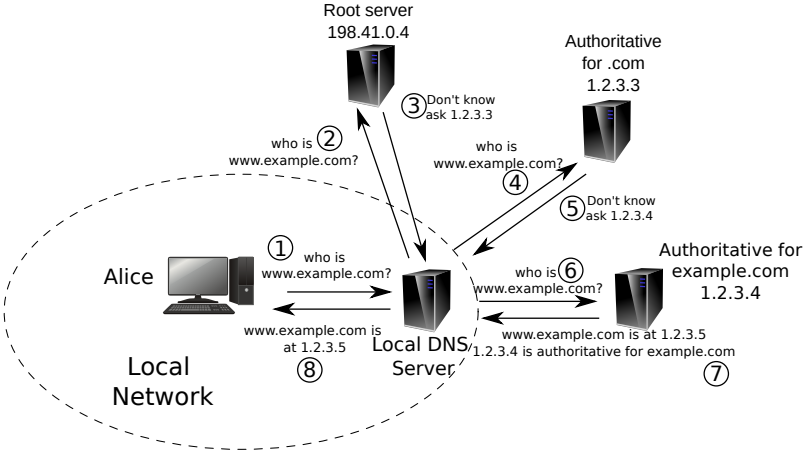
1. The browser of Alice interrogates the local DNS server asking for the address of `www.example.com`

DNS query



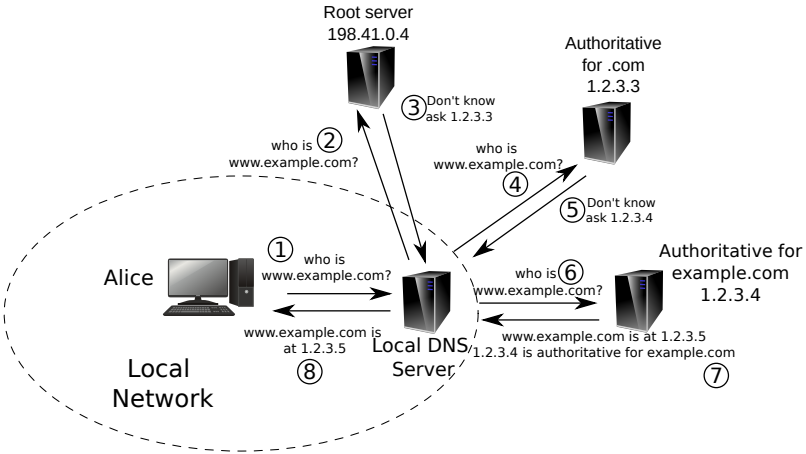
2. The local DNS server does not know the answer, so it queries one root DNS server

DNS query



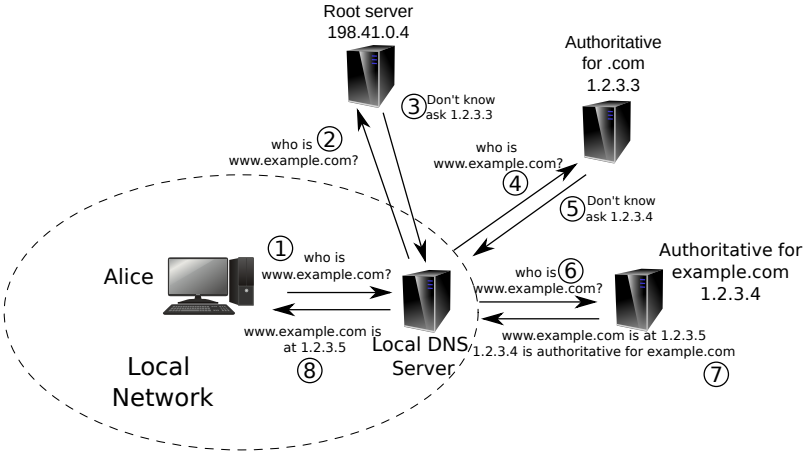
3. The root DNS server does not know the answer, but answers that 1.2.3.3 is the DNS server authoritative for `.com`

DNS query



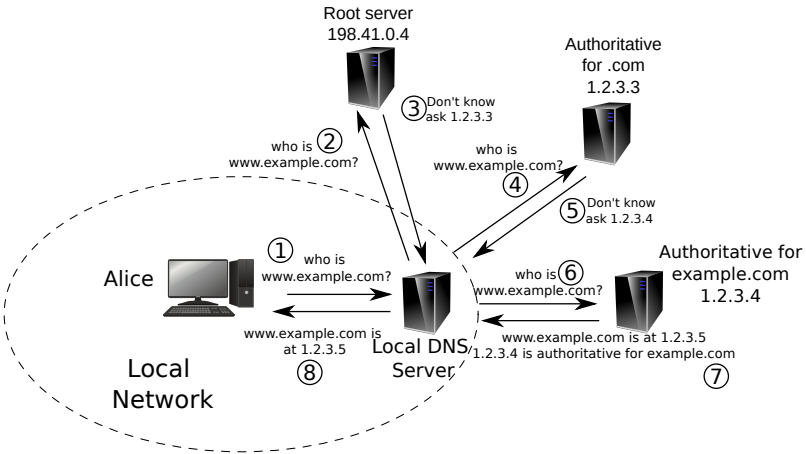
4. The local DNS server queries 1.2.3.3 for the "`.com`" top level domain (TLD)

DNS query



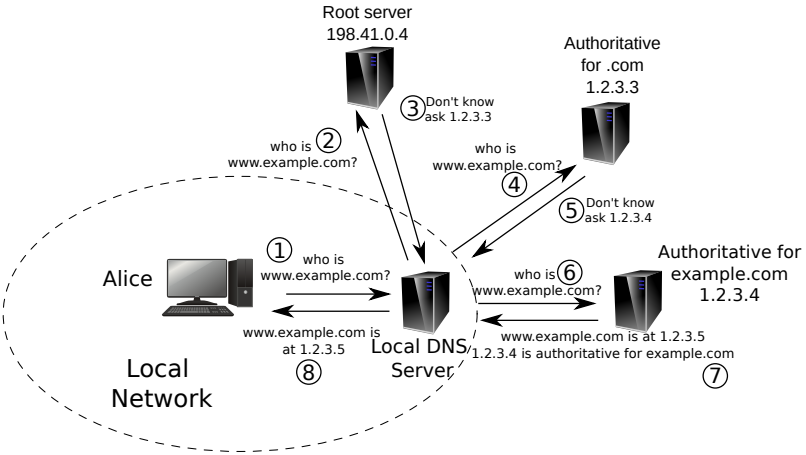
5. 1.2.3.3 does not know the answer, but answers that 1.2.3.4 is the DNS server authoritative for `example.com`

DNS query



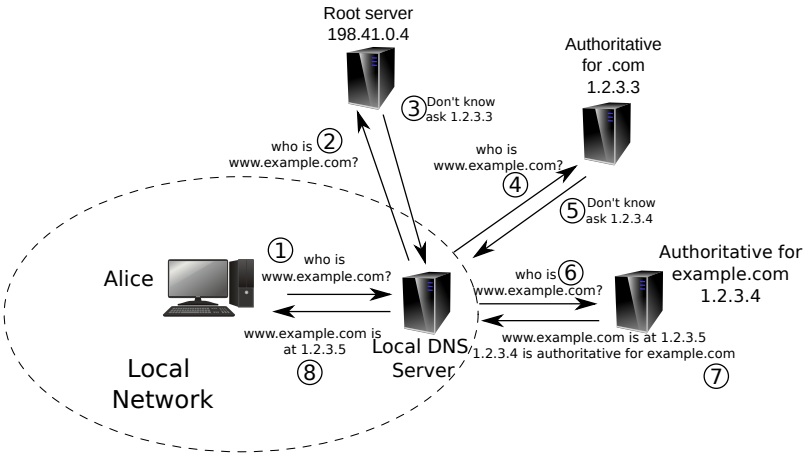
6. The local DNS server then queries 1.2.3.4

DNS query



7. The authoritative DNS server answers that 1.2.3.5 is the IP of `www.example.com`, and confirms its authority on `example.com`. Any further request for subdomains are to be sent to 1.2.3.4.

DNS query



8. The local DNS server sends the response to Alice

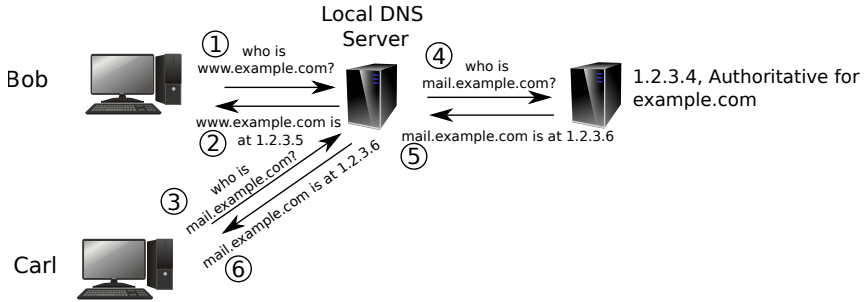
Recursive Vs Iterative

- The local DNS server receives what is called a **recursive** query
- It means that Alice is delegating the whole search to the server.
- The server can not return a partial result to Alice, it will keep making queries till when it finds the complete answer
- An **iterative** query instead can receive a partial answer.
- The root server, and the server authoritative for `.com` do not allow recursive queries, they allow iterative queries that provide answers with partial results.

Recursive Vs Iterative

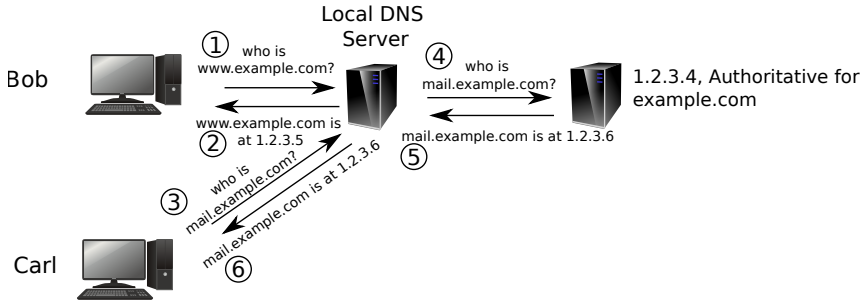
- Your local DNS instead allows recursive queries when asked from the local network, but not from outside of it.
- This is a typical behaviour, every DNS server makes recursive queries *only if it received a query from some host of the local network*
- For example, since the local DNS is not part of the local network of the `.com` authoritative server, the latter will not make a recursive query on behalf of the former.

DNS Caching



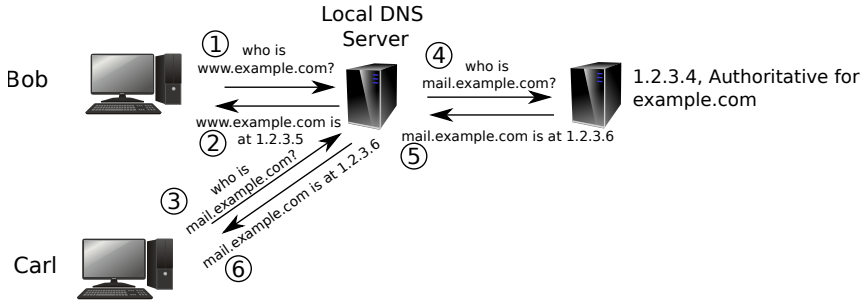
1. Later on, the browser of Bob interrogates the local DNS server asking for the address of `www.example.com`

DNS Caching



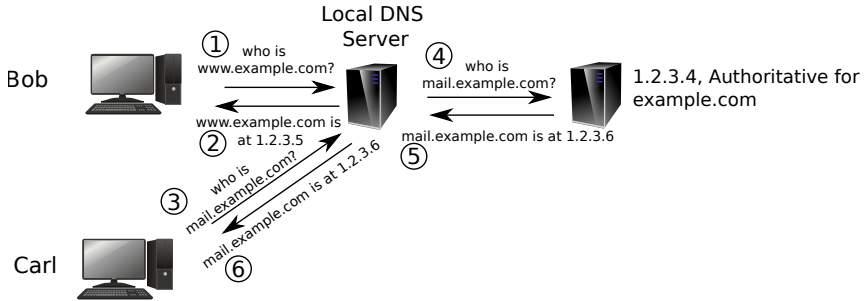
2. Now the local DNS has the answer in the cache, and can answer directly, saving time.

DNS Caching



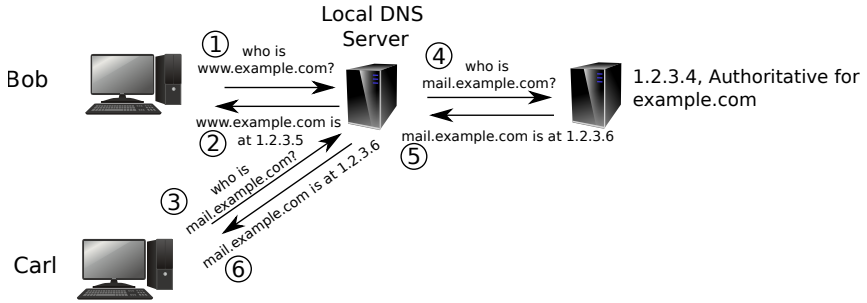
3. The browser of Carl queries `mail.example.com`, which is a different domain, so the local DNS does not know the answer

DNS Caching



4. However, the local DNS server then interrogates **directly** 1.2.3.4, that is authoritative for all `example.com` sub-domains. This saves time compared to interrogating the TLD DNS again. That's why an answer contains the `AUTHORITATIVE` section.

DNS Caching



5. The authoritative DNS server answers that 1.2.3.6 is the IP of `mail.example.com`
6. The local DNS server sends the response to Carl
7. Information is cached for a time included in the response

The Application Layer

↳1.2 DNS Protocol Detail

First or all, a couple of questions:

- Does the DNS service need a connection-oriented transport, or a connectionless one would be OK?

First or all, a couple of questions:

- Does the DNS service need a connection-oriented transport, or a connectionless one would be OK?
- Do you think that the DNS service is sensitive enough to require security services?

First or all, a couple of questions:

- Does the DNS service need a connection-oriented transport, or a connectionless one would be OK?
- Do you think that the DNS service is sensitive enough to require security services?
- And since the answer to the previous question is obviously **yes**, what would you consider the essential security services for the DNS service?

First or all, a couple of questions:

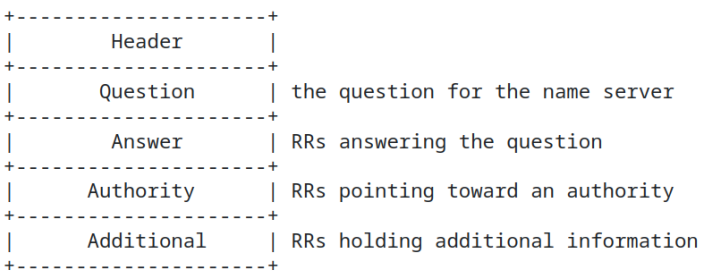
- Does the DNS service need a connection-oriented transport, or a connectionless one would be OK?
- Do you think that the DNS service is sensitive enough to require security services?
- And since the answer to the previous question is obviously **yes**, what would you consider the essential security services for the DNS service?

Whatever answer you gave to the previous question, the truth is, DNS was not designed with any security services in mind, and we still use it mostly like that

The DNS Protocol

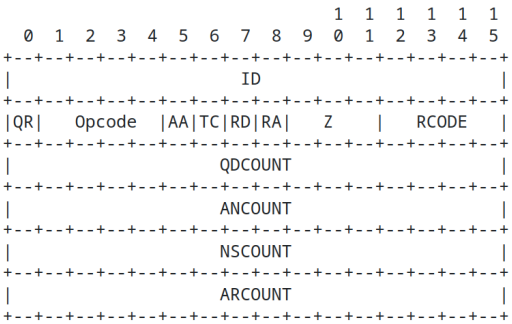
- RFC 1035 specifies the core of the DNS protocol: `https://datatracker.ietf.org/doc/html/rfc1035`
- The original DNS uses the connectionless UDP transport and a client-server architecture. Recent ones can use TCP, but the UDP version is still very much used.
- The client sends single messages and receives responses
- Let's see the format of a DNS message

Message



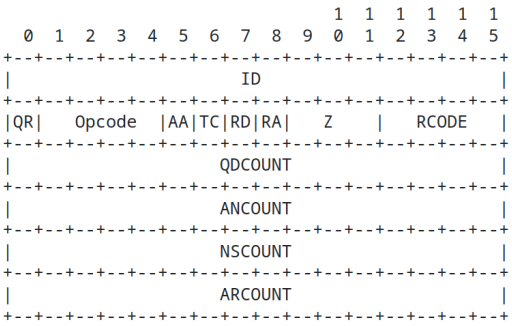
- A header whose format we will see
- A space for the eventual question
- A space for the answers
- A space for the authority fields
- Space for any other information.

Header Fields



- The Transaction `ID`: this is a unique number that the client inserts in the message, the server will replicate it in the answer
- The Transaction `ID` is meant to match the request with the reply in case the client does more than one request in parallel, for more than one domain
- Its choice must be random or it introduces security issues

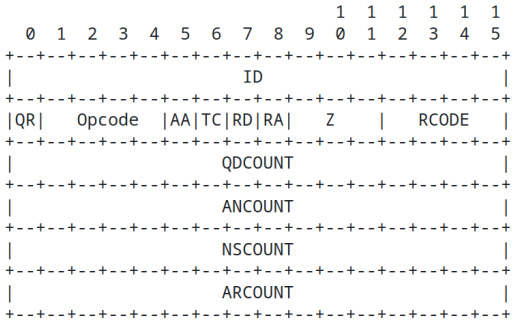
Header Fields



Header Flags:

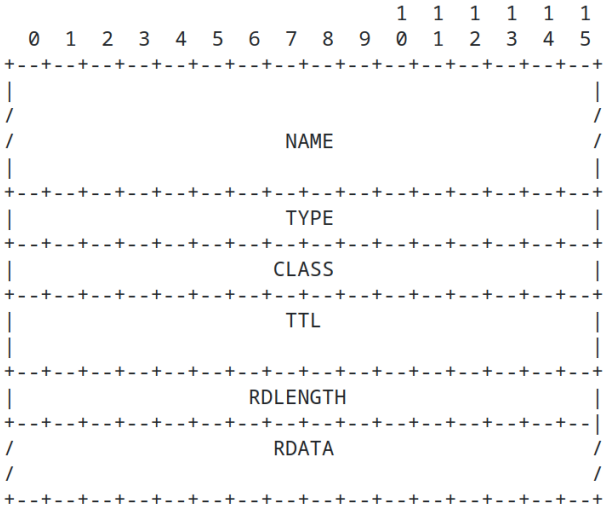
- `QR`: Query type, request (0), response (1)
- `AA`: Wether the server answering is *authoritative* (1) or not (0)
- `RD`: The client requests a *recursive* request (1)
- `RA`: Wether the server supports recursion (for future queries)
- `Z`: unused, reserved for future use
- `RCODE`: Response code, OK (0), errors (1-5)

Header Fields

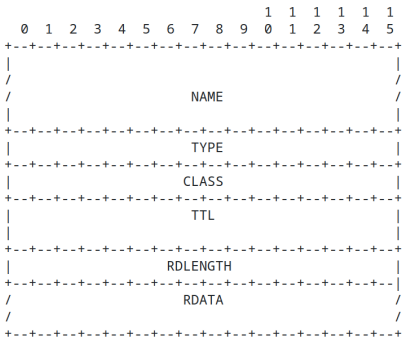


The following 4 fields express the number of Questions, Answers, Authority and Additional information in the rest of the message. Each of these items are called Resource Records (RR) and have their own format, I omitted the discussion of some fields for brevity.

Resource Records

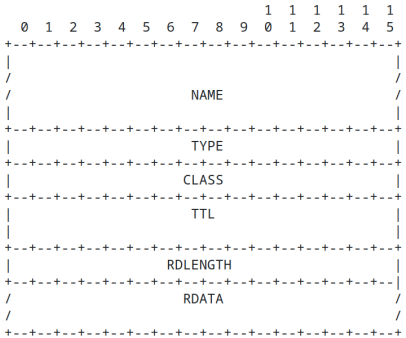


Resource Records



- `Name`: The name that was requested (both in the query and in the replies).
- Type:
  - `A`: IPv4 host address
  - `AAAA`: IPv6 host address
  - `NS`: Authoritative Name Server
  - `MX`: Mail Exchange
  - `CNAME`: Alias for the same resource
  - `SOA`: Generic data on the owner (e-mail, expiration...)
  - `TXT`: Anything

Resource Records



- `TTL`: The time this information can be cached, before it must be queried again
- `RDLENGTH`: The length of the `RDATA` field
- `RDATA`: The actual information

Extensible Protocols

- We have mentioned that at the lower layers of the stack, the headers are of fixed length and structure
- This makes it easier to parse by hardware (AKA: shift 64 left, XOR with 0x12a4, compare...)
- This makes it faster to elaborate and guarantees Mpackets/s speed.

Extensible Protocols

- However it makes it hard to extend/modify them. That's why often you will find bits that are *reserved for future use*[5]
- At the application layer the messages are elaborated by software so we don't expect that speed, and we can use more complex formats.

[5] Check the so-called Evil Bit `https://en.wikipedia.org/wiki/Evil_bit`

Extending DNS

- For instance, the fact you can have textual RR has allowed to introduce custom semantics, used to reduce spam and email forgery
- A TXT record like the following :
  `"v=spf1 ip4:192.0.2.0/24 ip4:198.51.100.123 a -all"`
  limits the IP addresses that can be used to send email using the a certain domain in the `Return-Path` field (more on this later on)

Reverse DNS

- When you register a domain, you can add a `PTR` RR, in which you specify a domain in a very special form, like:
  `5.3.2.1.in-addr.arpa`
- The `in-addr.arpa` domain is used to make so-called *reverse lookups*, that enable people to translate IP addresses to domain name (the opposite of DNS)
- When Alice wants to know if there is a domain associated to `1.2.3.5`, she will make a DNS query to search for `5.3.2.1.in-addr.arpa`, and this will provide the list of domains associated to it (note the inversion of numbers).
- `PTR` RR are not mandatory, it is optional to add them and make your IP address searchable in reverse queries.
- The process is not trivial: see this document

Why reversing the bytes?

- Because also the reverse DNS is hierarchical. `in-addr.arpa` does not know all the resolutions, but it delegates to those entities that own the IP addresses
- and since IP addresses are hierarchical, we need to follow the hierarchy

First let's find out the DNS servers at `in-addr.arpa`

```
$ dig in-addr.arpa  NS
;; ANSWER SECTION:
in-addr.arpa.    2611  IN  NS  e.in-addr-servers.arpa.
in-addr.arpa.    2611  IN  NS  f.in-addr-servers.arpa.
in-addr.arpa.    2611  IN  NS  a.in-addr-servers.arpa.
in-addr.arpa.    2611  IN  NS  d.in-addr-servers.arpa.
in-addr.arpa.    2611  IN  NS  c.in-addr-servers.arpa.
in-addr.arpa.    2611  IN  NS  b.in-addr-servers.arpa.
```

Now let's ask to one of them about 157.138.7.88 (@ specifiy the target DNS server)

```
dig @a.in-addr-servers.arpa 88.7.138.157.in-addr.arpa PTR
;; ANSWER SECTION:
157.in-addr.arpa. 86400 IN  NS  z.arin.net.
157.in-addr.arpa. 86400 IN  NS  u.arin.net.
157.in-addr.arpa. 86400 IN  NS  r.arin.net.
157.in-addr.arpa. 86400 IN  NS  x.arin.net.
157.in-addr.arpa. 86400 IN  NS  y.arin.net.
157.in-addr.arpa. 86400 IN  NS  arin.authdns.ripe.net.
```

The answer is that ARIN (`z.arin.net`) is responsible

Let's ask to `z.arin.net`

```
$ dig @z.arin.net 88.7.138.157.in-addr.arpa PTR
;; AUTHORITY SECTION:
7.138.157.in-addr.arpa. 86400 IN  NS  algol.unive.it.
7.138.157.in-addr.arpa. 86400 IN  NS  dns.cineca.it.
7.138.157.in-addr.arpa. 86400 IN  NS  ns1.garr.net.
7.138.157.in-addr.arpa. 86400 IN  NS  vega.unive.it.

```

`algol.unive.it` is authoritative

```
$ dig @algol.unive.it. 88.7.138.157.in-addr.arpa PTR
;; ANSWER SECTION:
88.7.138.157.in-addr.arpa. 86400 IN PTR www.unive.it.
```

The answer is (`www.unive.it`) is responsible

WHOIS protocol

- The WHOIS protocol (now updated by the RDAP protocol, but we still refer to it as WHOIS) allows to access the information associated with a certain domain
- a WHOIS query asks to the regional register (RIR) to provide the address of the Registrar responsible for a domain, that is then queried to provide all the data that were used for the registration (Owner, contact e-mail, expiry date...)

In Practice

- Let's make some practical examples.
- On GNU/Linux the commands used to access DNS data are:
  - `host`: make a DNS query or a reverse DNS query
  - `whois`: make a `WHOIS` query
  - `wireshark`: a generic packet dissector that can be used to analyse traffic

In Practice

Examples:

- `host -v unive.it`
- `host -v`
- `whois`: make a `WHOIS` query
- `wireshark`: a generic packet dissector that can be used to analyse traffic

```
$ host -v unive.it
Trying "unive.it"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47477
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;unive.it.      IN  A

;; ANSWER SECTION:
unive.it.   5802  IN  A 157.138.7.88

Received 42 bytes from 127.0.0.53#53 in 0 ms
Trying "unive.it"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6729
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;unive.it.      IN  AAAA

Received 26 bytes from 127.0.0.53#53 in 4 ms
Trying "unive.it"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57474
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;unive.it.      IN  MX

;; ANSWER SECTION:
unive.it.   7180  IN  MX  10 ASPMX3.GOOGLEMAIL.COM.
unive.it.   7180  IN  MX  10 ASPMX2.GOOGLEMAIL.COM.
unive.it.   7180  IN  MX  5 ALT2.ASPMX.L.GOOGLE.COM.
unive.it.   7180  IN  MX  5 ALT1.ASPMX.L.GOOGLE.COM.
unive.it.   7180  IN  MX  1 ASPMX.L.GOOGLE.COM.
Received 159 bytes from 127.0.0.53#53 in 0 ms
```

```
$ host -v 157.138.7.88
Trying "88.7.138.157.in-addr.arpa"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10957
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;88.7.138.157.in-addr.arpa. IN  PTR

;; ANSWER SECTION:
88.7.138.157.in-addr.arpa. 86400 IN PTR www.unive.it.

Received 69 bytes from 127.0.0.53#53 in 80 ms
```

```
$ whois unive.it
Domain:             unive.it
Status:             ok
Signed:             no
Created:            1996-01-29 00:00:00
Last Update:        2023-02-14 01:04:59
Expire Date:        2024-01-29

Registrant
  Organization:     Universita degli Studi di Venezia
  Address:          Area Servizi Informatici e Telecomunicazioni
                    Venezia
                    30123
                    VE
                    IT
  Created:          2007-03-01 10:27:43
  Last Update:      2022-01-28 11:20:26

Admin Contact
  Name:             hidden
  Organization:     hidden

Technical Contacts
  Name:             Alvise Rabitti
  Organization:     Universita degli Studi di Venezia
  Address:          Area Servizi Informatici e Telecomunicazioni
                    Venezia
                    30123
                    VE
                    IT
  Created:          2022-01-28 11:14:40
  Last Update:      2022-01-28 11:14:40
```

```
1
2   Name:              Stefano Claut
3   Organization:      Universita di Venezia
4   Address:           Universita Ca Foscari Dorsoduro 3861
5                      Venezia
6                      30123
7                      VE
8                      IT
9   Created:           2022-10-05 11:12:53
10  Last Update:       2022-10-05 11:12:53
11
12 Registrar
13  Organization:      Consortium GARR
14  Name:              GARR-REG
15  Web:               http://www.garr.it
16  DNSSEC:            no
17
18
19 Nameservers
20  algol.unive.it
21  vega.unive.it
22  dns.cineca.it
23  ns1.garr.net
```

The Application Layer

↳1.3 DNS Operation

Server Redundancy

- Normally, during a resolution, the local DNS server discovers more than one authoritative DNS server
- DNS server are redundant. If one does not answer, the same query is repeated in a round-robin way to all the others. Example:

```
$ dig google.com

;; ANSWER SECTION:
google.com.    97  IN  A 216.58.204.142

;; AUTHORITY SECTION:
google.com.    97  IN  NS  ns2.google.com.
google.com.    97  IN  NS  ns1.google.com.
google.com.    97  IN  NS  ns3.google.com.
google.com.    97  IN  NS  ns4.google.com.

```

- Google returns one IP for the A record and 4 DNS servers for the NS records.
- All of them should work, and if one doesn't you can use the other ones
- remember it's UDP, you can't distinguish if a packet is lost or the server is just down, so you need back-ups

Round Robin DNS

- In some cases, if you query the authoritative DNS twice in a row, it will answer with two different IP addresses, even before the TTL expires.
- This is because the same service (i.e. a website) can be provided by more than one server
- In practice, there is a replica of the same service in N different machines.

Round Robin DNS

- The DNS server will return one of the IP in a round-robin way
- So the clients will connect not always to the same server, but to one of the available ones
- This is done to enforce some load balancing among the servers.

Importance of TTL

- TTL is a field of the DNS responses.
- TTL determines for how long a DNS server can keep the information obtained from another DNS server
- There are two extremes:
  - If TTL is very long, the caches are valid for a long time, and less DNS requests are received
  - If TTL is very short, any change due to whatever reason is propagated fast

Try it yourself

Multiple `dig`, with edited output. Second column is the TTL.

```
1  $ dig google.com
2  ;; ANSWER SECTION:
3  google.com.   70  IN  A 216.58.204.142
4
5  $ sleep 10; dig google.com
6  ;; ANSWER SECTION:
7  google.com.   59  IN  A 216.58.204.142
8
9  $sleep 55; dig google.com
10 ;; ANSWER SECTION:
11 google.com.   5 IN  A 216.58.204.142
12
13 $sleep 5; dig google.com
14 ;; ANSWER SECTION:
15 google.com.   3183  IN  A 216.58.204.142
```

line 3-11: My resolver is querying the local DNS, and it receives cached data with `TTL` from 70 to 5

line 15: `TTL` expires. It means I triggered a new request from my local DNS, the cache was refreshed to 3183 seconds

DNS Hard Facts

- Names do not imply location: `example.it` can be hosted anywhere in the world.
- There can be more than one DNS server with the same IP address. This is called IP `anycast` and we will explain it when talking about BGP.
- Some record types are **non-terminal**, that means that even if you reached the authoritative server, it does not reply with an IP but with another domain (see `NS` replies in previous examples). You then need another query.

DNS Exercises

- You can now make all the exercises on `inginious.org` for the DNS protocol
- The only caveat, some questions refer to DNS over TCP, that is a less used variant.
- Refer to RFC 1035 to check the message format for DNS over TCP and answer the question

Sect. 2 Extra Slides: DNS Security

A note on Security

- DNS is a critical service under the security point of view
- If an attacker can modify the responses of a DNS query, it can have nefarious consequences on the users privacy.

Domain Hijacking

- At the base of the security issues there is an attacker that can modify the DNS response and make Alice believe that the domain she wants to browse (`www.bob.com`) corresponds to the attacker IP
- This could be done by:
  - Someone in the middle: typically for content censorship
  - Someone who hacked the DNS server
  - Someone who can spoof a correct response from a DNS server
- Since there is no cryptographically secure association between name and IP address these things are all possible.
- Let's see case by case

Domain Censorship

- One typical application is the censorship of certain domains
- An Internet connected PC has been configured to use a certain local DNS server. This is typically provided by the Internet Service Provider (ISP)
- The ISP will receive all the DNS queries, and can decide to answer to some of them with a fake reply
- Typical reasons are:
  - State-censorship: some content is deemed inappropriate by the government. An example is a the so-called *great firewall of China*
  - Application of local or international law: the typical example is the application of a court ruling that forbids a certain website to be navigated.
  - In some cases the ISP hijacks not existing domains to show advertisement, this is a very bad practice.

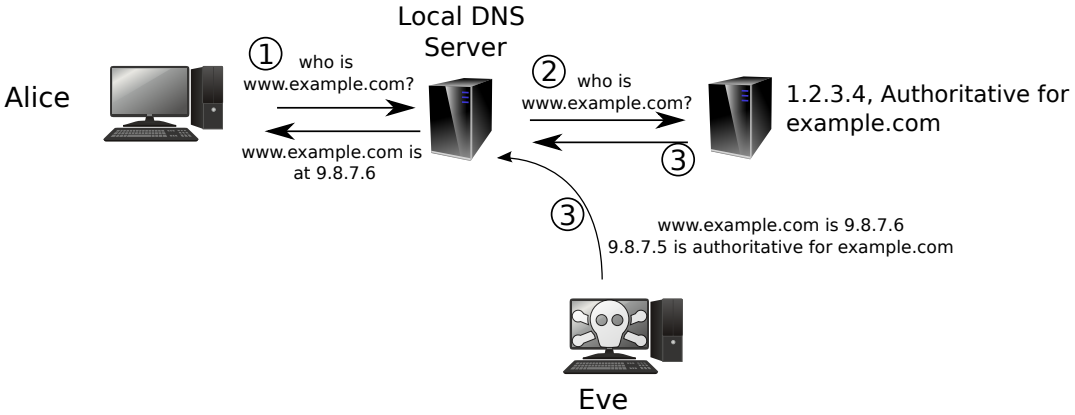
DNS Vulnerability

- In some cases the DNS server itself was hacked, and the contents were changed
- This could happen as a consequence of a technical attack, that means that someone exploited a vulnerability in the software
- Or it could happen as a consequence of social engineering, that is, someone was able to convince the DNS administrator to change the IP impersonating the owner.
- In the dumbest cases, the owner forgot to renew the domain and someone else bought it.

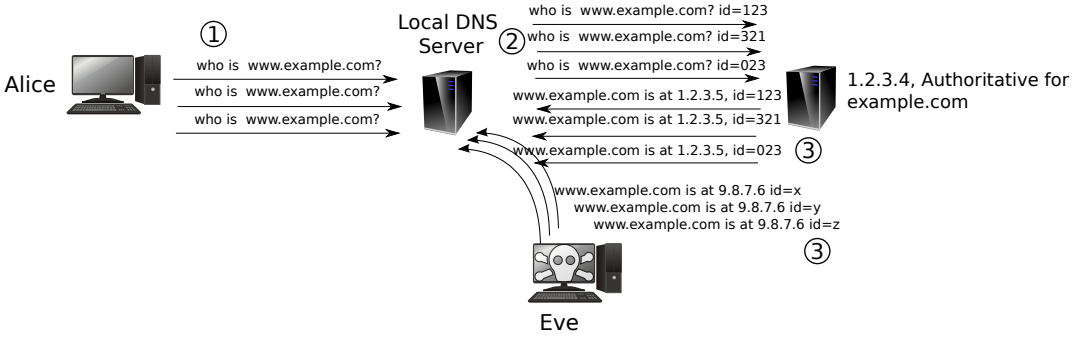
Spoofing the Response

- If the attacker is a MiTM, then spoofing a response is a very easy thing to do
- However, before the introduction of specific extensions, it was possible for an attacker to try to poison a DNS response without being in the middle.

Kaminksy Attack



Kaminksy Attack



Kaminsky Attack: a Sketch

- In the Kaminsky attack Alice cooperates with Eve (or better said, Eve was able to hack into a computer in the network of the DNS server)
- Alice generates a request for `mail.example.com`
- Now the local DNS server knows the authoritative server for `example.com`

Kaminsky Attack: a Sketch

- Alice then generates a X requests for `www.example.com`, each one uses a different ID
- The local DNS server will generate X parallel requests for `1.2.3.4`
- `1.2.3.4` will produce X reply, with roughly the same TTL
- Each reply carries the ID of a request
- The first one that arrives will trigger the response to Alice, and will create an entry in the cache

Eve Forges Spoofed Replies

- Eve must forge X replies, trying to guess the ID
- Her hope is that among her replies there is one that matches one of the ID generated by the local DNS server, and that all her replies arrive before the original one
- If this happens, the cache is poisoned for a TTL time she adds to the reply
- Note that the reply can also include an `NS` record, so that the Eve can set-up her own DNS server and trick the local DNS server to convince it that Eve's server is authoritative for a whole domain.

Eve Forges Spoofed Replies

- The larger is the amount of random bits to be guessed, the lower the probability of success.
- With 32 random bits, Eve needs about 140.000 packets to be almost sure to have success.
- This is not a small number, neither an impossible one.

DNS Denial of Service

- As an example of a DoS, consider the case in which Eve forges a high number of DNS requests using the IP address of Bob
- DNS is a stateless protocol, the DNS server will just answer to Bob
- The DNS request can be as small as a few tens of bytes
- The DNS reply can be big, as it may include information for many different records. Some servers support the `ANY` record, that will provide all the information they own.
- It easy to check with dig that a simple request (about 80B) can produce a large reply (even more than ten times larger, try: `dig unive.it ANY`)

DNS DoS

- The ingredients for a reflection attack are present:
  - The protocol is stateless
  - There is a relevant amplification factor N
- The attacker can then spoof a large number of DNS requests using Bob source address, and Bob will receive $\times N$ traffic that can saturate its own network resources

Extra Slides: DNS Security

↳2.1 DNS Cookies

DNS Cookies

- RFC 7873 introduced so-called DNS cookies that make this kind of attack harder
- The principle of operation is pretty simple:
  - The first time that a DNS server queries another, it adds to the message an optional `RR` named `DNS COOKIE`. This contains the DNS cookie, that is a randomly generated number (8 Bytes).
  - When replying to a query, the server will generate a cookie itself, and add it in the reply (8 Bytes at least).
  - From that moment on, all queries must include both cookies.

DNS Cookies

- If a server receives a query without a cookie, it is free to treat the query as a suspicious one.
- One typical consequence is to rate-limit the queries from client that do not provide valid cookies.
- Rate-limit means to accept only a limited amount of queries per second and drop the extra ones.
- This mitigates both the attacks:
  - It adds more random material to be guessed: $2^{64+64+32}$ is too long to have a reasonable probability of being guessed
  - It provides a way to stop the flood of DNS requests, that is at the base of both attacks.

DNS Cookies: Exercise

In the real specification there are some subtleties I don't address:

- The cookies are not *just* random numbers, they are generated as a function of some input parameter.
- Read the relevant part of the RFC (4.1, 4.2)
  `https://datatracker.ietf.org/doc/html/rfc7873#section-4.1`
- Answer the question, why is a cookie created that way, and not just as a random number?