# Analisando Tráfego de Redes e *Sniffing* de Pacotes

Prof. Ricardo Mesquita

# Capturando e Injetando Pacotes com pcapy-ng

- Para instalar o pcapy-ng em seu sistema operacional, você pode usar o seguinte comando:

```
$ pip install pcapy-ng
Collecting pcapy-ng
  Downloading pcapy-ng-1.0.9.tar.gz (38 kB)
Building wheels for collected packages: pcapy-ng
  Building wheel for pcapy-ng (setup.py) ... done
  Created wheel for pcapy-ng: filename=pcapy_ng-1.0.9-cp310-cp310-linux_x86_64.whl
size=79955 sha256=3b2e321d2bc02106c1d3899663f9d5138bb523397c246274501cdc1c74f639e9
  Stored in directory:
/root/.cache/pip/wheels/27/de/8d/474edb046464fd3c3bf2a79dec3222b732b5410d2e0097d2b0
Successfully built pcapy-ng Installing collected packages: pcapy-ng
Successfully installed pcapy-ng-1.0.9
```

# Capturando Pacotes com pcapy-ng

- O módulo pcapy-ng fornece o método **open_live()** para capturar pacotes de uma interface específica.
- Podemos especificar o número de bytes por captura e outros parâmetros como modo promíscuo e tempo limite.
- No exemplo a seguir, usamos o método **findalldevs()** para obter todas as interfaces da sua máquina e obtemos os bytes capturados usando a interface selecionada.

# Capturando Pacotes com pcapy-ng

```
import pcapy
import datetime
interfaces = pcapy.findalldevs()
print("Available interfaces are :")
for interface in interfaces:
        print(interface)
interface = input("Enter interface name to sniff : ")
print("Sniffing interface " + interface)
cap = pcapy.open_live(interface, 65536 , 1 , 0)
while True:
        (header, payload) = cap.next()
        print ('%s: captured %d bytes' %(datetime.datetime.now(),
                                                header.getlen()))
```

# Capturando Pacotes com pcapy-ng

- Você pode selecionar uma interface de rede de interesse.
- Invoque o script novamente, desta vez usando privilégios sudo e veremos os bytes capturados na interface em tempo real:

```
$ sudo python pcapy_capturing_packets.py
Available interfaces are:
wlo1
any
lo
....
Enter interface name to sniff: wlo1
Sniffing interface wlo1
2022-12-03 17:39:09.033355: captured 412 bytes
2022-12-03 17:39:09.033435: captured 432 bytes
2022-12-03 17:39:09.033492: captured 131 bytes ...
```

**Observação**:
Note que normalmente precisaremos executar os comandos com sudo, pois o acesso às interfaces requer acesso de administrador do sistema.

# Lendo Cabeçalhos de Pacotes

- No exemplo a seguir, capturamos pacotes de um dispositivo específico (wlo1) e, para cada pacote, obtemos o cabeçalho e a carga útil para extrair informações sobre endereços MAC, cabeçalhos IP e protocolo.

# Lendo Cabeçalhos de Pacotes

```
import pcapy
from struct import *
interfaces = pcapy.findalldevs()
print("Available interfaces are :")
for interface in interfaces:
        print(interface)
interface = input("Enter interface name to sniff : ")
cap = pcapy.open_live(interface, 65536, 1, 0)
while True:
        (header,payload) = cap.next()
        l2hdr = payload[:14]
        l2data = unpack("!6s6sH", l2hdr)
        srcmac = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (l2hdr[0], l2hdr[1], l2hdr[2], l2hdr[3], l2hdr[4], l2hdr[5])
        dstmac = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (l2hdr[6], l2hdr[7], l2hdr[8], l2hdr[9], l2hdr[10],l2hdr[11])
        print("Source MAC: ", srcmac, " Destination MAC: ", dstmac)
        ipheader = unpack('!BBHHHBBH4s4s' , payload[14:34])
        timetolive = ipheader[5]
        protocol = ipheader[6]
        print("Protocol ", str(protocol), " Time To Live: ", str(timetolive))
```

```
$ sudo python pcapy_reading_headers.py
Available interfaces are :
wlo1
any
lo
enp0s25
docker0
br-9ab711bca770
bluetooth0
bluetooth-monitor
nflog
nfqueue
dbus-system
dbus-session
Enter interface name to sniff :wol1
Source MAC: a4:4e:31:d8:c2:80 Destination MAC: f4:1d:6b:dd:14:d0
Protocol 6 Time To Live: 234
Source MAC: f4:1d:6b:dd:14:d0 Destination MAC: a4:4e:31:d8:c2:80
Protocol 6 Time To Live: 64 …..
```

**Note:**
Ao executar o script anterior, ele retorna os endereços MAC e o tempo de vida de cada um dos pacotes capturados.

# Lendo Arquivos pcap com pcapy-ng

- No processo de captura de pacotes é comum encontrar arquivos com extensão **.pcap**.
- Este arquivo contém frames e pacotes de rede e é muito útil se precisarmos salvar o resultado de uma análise de rede para processamento posterior.
- As informações armazenadas em um arquivo **.pcap** podem ser analisadas quantas vezes forem necessárias sem que o arquivo original seja alterado.
- Com a função **open_offline()**, podemos ler um arquivo **pcap** e obter uma lista de pacotes que podem ser manipulados diretamente pelo Python.

```
import pcapy
from struct import *
pcap_file = pcapy.open_offline("packets.pcap")
count = 1
while count<500:
        print("Packet #: ", count)
        count = count + 1
        (header,payload) = pcap_file.next()
        l2hdr = payload[:14]
        l2data = unpack("!6s6sH", l2hdr)
        srcmac = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (l2hdr[0], l2hdr[1], l2hdr[2], l2hdr[3],
                                                    l2hdr[4], l2hdr[5])
        dstmac = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (l2hdr[6], l2hdr[7], l2hdr[8], l2hdr[9],
                                                    l2hdr[10],l2hdr[11])
        print("Source MAC: ", srcmac, " Destination MAC: ", dstmac)
        ipheader = unpack('!BBHHHBBH4s4s' , payload[14:34])
        timetolive = ipheader[5]
        protocol = ipheader[6]
        print("Protocol ", str(protocol), " Time To Live: ", str(timetolive))
        count = count + 1
```

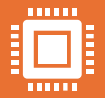
**Note**:

- Lemos os primeiros 500 pacotes na captura packets.pcap incluída na pasta pcapy.
- Para cada pacote, obtemos os endereços MAC de origem e destino, bem como o protocolo e o TTL do pacote.

# Capturando e Injetando Pacotes com scapy

- A análise do tráfego de rede (os pacotes que são trocados entre dois hosts), pode nos ajudar a identificar os detalhes dos sistemas que participam da comunicação.
- A mensagem e a duração da comunicação são algumas das informações valiosas que um invasor que está escutando no meio da rede pode obter.

# Uma Introdução ao scapy



scapy é um módulo escrito em Python para manipular pacotes de dados com suporte a múltiplos protocolos de rede.

Permite a criação e a modificação de pacotes de rede de vários tipos.

Implementa funções para capturar e detectar pacotes passivamente e, em seguida, executa ações nesses pacotes.

Embora originalmente projetado para o Linux, a versão mais recente do scapy oferece suporte ao Windows.

# Uma Introdução ao scapy

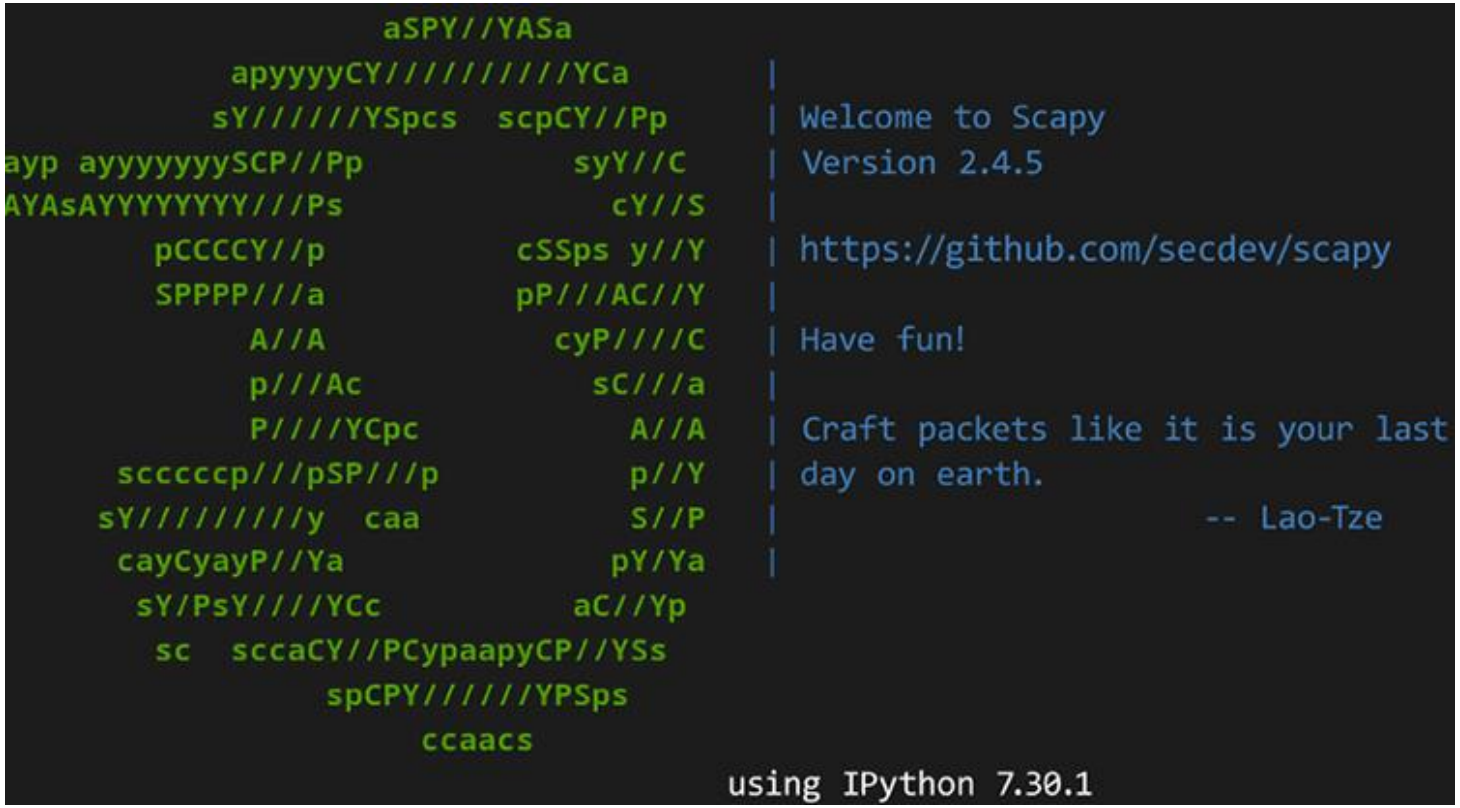
- Para instalar o scapy, você pode seguir as instruções em https://scapy.net e executar o seguinte comando:

```
$ sudo pip install scapy
Collecting scapy
   Downloading scapy-2.4.5.tar.gz (1.1 MB)
        |████████████████████████████████| 1.1 MB 4.6 MB/s
Building wheels for collected packages: scapy
   Building wheel for scapy (setup.py) ... done
   Created wheel for scapy: filename=scapy-2.4.5-py2.py3-none-any.whl
size=1261554sha256=15d3e4d36f73cdf2fd319ee17047d49cba49ae0a14e7ad90556784247f220f84
   Stored in directory:
/root/.cache/pip/wheels/85/7a/e6/48f944c02302d8d0252c148bdab7616a1567737c1e57117c31
Successfully built scapy
   Installing collected packages: scapy
   Successfully installed scapy-2.4.5
```

# Uma Introdução ao scapy

- Ao instalar o scapy em seu sistema operacional, você pode acessar sua interface de linha de comando (CLI) da seguinte maneira:

```
$ scapy
```



# Comandos do scapy

- Podemos usar o scapy de duas maneiras: interativamente em uma janela de terminal ou a partir de um script Python, importando-o como uma biblioteca.
- As principais funções que podemos usar são:

  - **ls()**: Lista das camadas disponíveis.
  - **explore()**: Interface gráfica para visualizar camadas existentes.
  - **lsc()**: Funções disponíveis.
  - **send()**: Envia pacotes para o nível 2.
  - **sendp()**: Envia pacotes para o nível 3.
  - **sr()**: Envia e recebe pacotes no nível 3.
  - **srp()**: Envia e recebe pacotes no nível 2.
  - **sr1()**: Envia e recebe apenas o primeiro pacote no nível 3.
  - **srp1()**: Envia e recebe apenas o primeiro pacote no nível 2.
  - **sniff()**: sniffing de pacotes.
  - **traceroute()**: comando traceroute.
  - **arping()**: Envio de solicitações ARP 'quem tem' para determinar quais hosts estão ativos na rede.

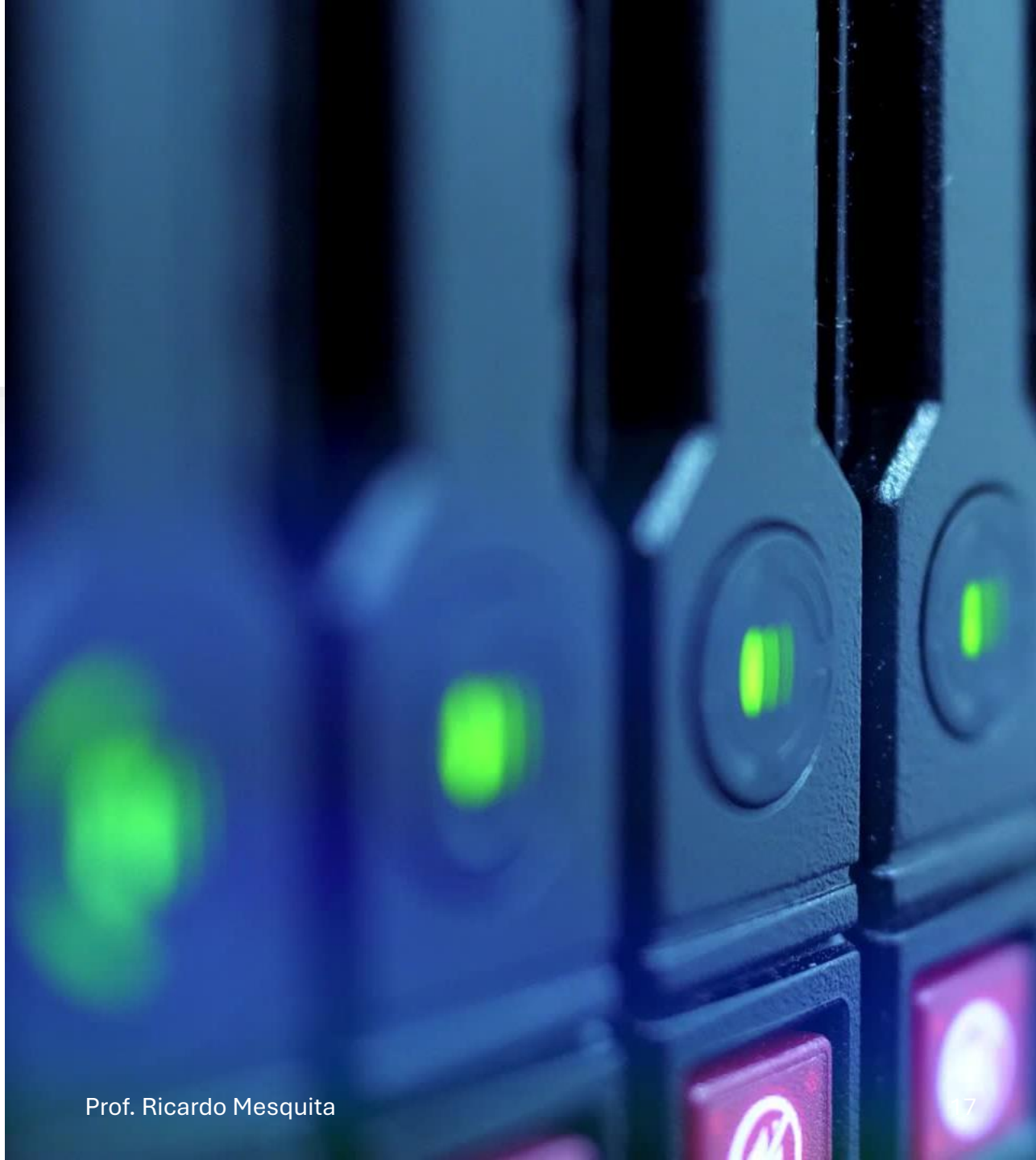
# Comandos do scapy

- O scapy suporta mais de 300 protocolos de rede. Podemos obter a lista de protocolos suportados pelo scapy usando o comando **ls()**:

```
>>> ls()
AH        : AH
AKMSuite  : AKM suite
ARP       : ARP
ASN1P_INTEGER : None
ASN1P_OID : None
ASN1P_PRIVSEQ : None
ASN1_Packet : None
ATT_Error_Response : Error Response
ATT_Exchange_MTU_Request : Exchange MTU Request
ATT_Exchange_MTU_Response : Exchange MTU Response
ATT_Execute_Write_Request : Execute Write Request
ATT_Execute_Write_Response : Execute Write Response
ATT_Find_By_Type_Value_Request : Find By Type Value Request
……..
```

# Comandos do scapy

- Podemos indicar a camada sobre a qual desejamos mais informações.
- A seguir mostramos uma execução do comando ls() com diferentes parâmetros, onde podemos ver os campos suportados pelos protocolos IP, ICMP e TCP.



```
>>> ls(IP)
version     : BitField (4 bits)     = ('4’)
ihl         : BitField (4 bits)     = ('None’)
tos         : XByteField                = ('0’)
len         : ShortField                = ('None’)
id          : ShortField                = ('1’)
flags       : FlagsField                = ('<Flag 0 ()>’)
frag        : BitField (13 bits)    = ('0’)
ttl         : ByteField                 = ('64’)
proto       : ByteEnumField         = ('0’)
chksum      : XShortField               = ('None’)
src         : SourceIPField         = ('None’)
dst         : DestIPField               = ('None’)
options     : PacketListField           = ('[]')
```

```
>>> ls(ICMP)
type            : ByteEnumField                         = ('8’)
code            : MultiEnumField (Depends on 8)         = ('0’)
chksum          : XShortField                           = ('None’)
id              : XShortField (Cond)                    = ('0’)
seq             : XShortField (Cond)                    = ('0’)
ts_ori          : ICMPTimeStampField (Cond)             = ('70780296’)
ts_rx           : ICMPTimeStampField (Cond)             = ('70780296’)
ts_tx           : ICMPTimeStampField (Cond)             = ('70780296’)
gw              : IPField (Cond)                        = ("'0.0.0.0’”)
ptr             : ByteField (Cond)                      = ('0’)
reserved        : ByteField (Cond)                      = ('0’)
length          : ByteField (Cond)                      = ('0’)
addr_mask       : IPField (Cond)                        = ("'0.0.0.0’”)
nexthopmtu      : ShortField (Cond)                     = ('0’)
unused          : MultipleTypeField (ShortField, IntField, StrFixedLenField) = ("b''")
```

```
>>> ls(TCP)
sport     : ShortEnumField          = ('20’)
dport     : ShortEnumField          = ('80’)
seq       : IntField                = ('0’)
ack       : IntField                = ('0’)
dataofs   : BitField (4 bits)       = ('None’)
reserved  : BitField (3 bits)       = ('0’)
flags     : FlagsField              = ('<Flag 2 (S)>’)
window    : ShortField              = ('8192’)
chksum    : XShortField             = ('None’)
urgptr    : ShortField              = ('0’)
options   : TCPOptionsField         = ("b''")
```

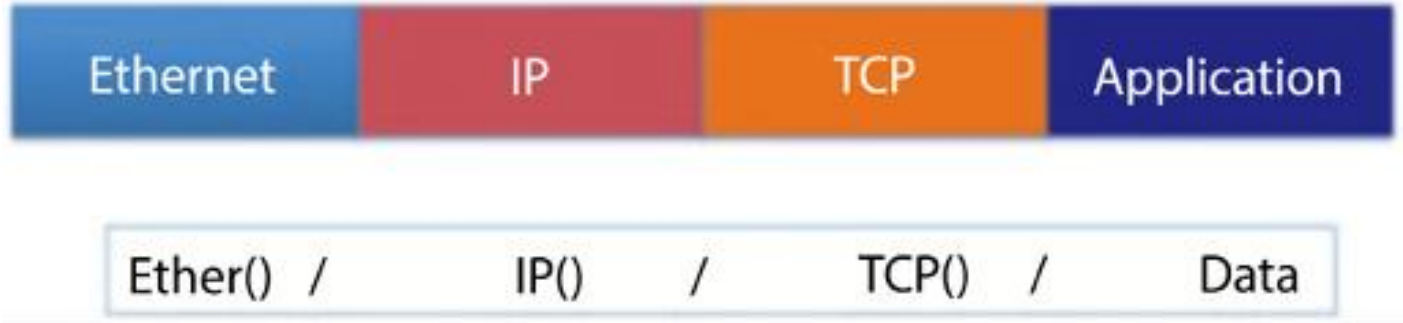
# Comandos do scapy

- Pode-se ver as funções disponíveis no scapy com o comando lsc():

```
>>> lsc()
IPID_count           : Identify IP id values classes in a list of packets
arpcachepoison       : Poison target's cache with (your MAC,victim's IP) couple
arping               : Send ARP who-has requests to determine which hosts are up
arpleak              : Exploit ARP leak flaws, like NetBSD-SA2017-002.
bind_layers          : Bind 2 layers on some specific fields' values.
bridge_and_sniff     : Forward traffic between interfaces if1 and if2, sniff and return
chexdump             : Build a per-byte hexadecimal representation
computeNIGroupAddr   : Compute the NI group Address. Can take a FQDN as the input parameter
```

# Comandos do scapy

- O scapy permite a criação de pacotes personalizados em qualquer uma das camadas do protocolo TCP/IP.
- Os pacotes são criados por camadas, começando pela camada mais baixa até chegar à camada de aplicação.
- No diagrama a seguir, podemos ver a estrutura que o scapy gerencia por camada.



# Comandos do scapy

- No scapy, uma camada geralmente representa um protocolo.
- Sabemos que:
  - Os protocolos de rede são estruturados em pilhas, onde cada etapa consiste em uma camada ou protocolo.
  - Um pacote de rede consiste em múltiplas camadas e cada camada é responsável por parte da comunicação.

# Comandos do scapy

Um pacote em scapy é um conjunto de dados estruturados prontos para serem enviados a uma rede.

Os pacotes devem seguir uma estrutura lógica, de acordo com o tipo de comunicação que se deseja simular.

Isso significa que se você quiser enviar um pacote TCP/IP, deverá seguir as regras do protocolo definidas no padrão TCP/IP.

# Comandos do scapy

- Por padrão, a camada IP é configurada como o IP de destino do endereço localhost em 127.0.0.1, que se refere à máquina local onde o scapy é executado.
- Poderíamos executar o scapy na linha de comando para verificar nosso endereço de host local:

```
>>> ip =IP()
>>> ip.show()
###[ IP ]###
version        = 4
ihl            = None
tos            = 0x0
len            = None
id             = 1
flags          =
frag           = 0
ttl            = 64
proto          = hopopt
chksum         = None
src            = 127.0.0.1
dst            = 127.0.0.1
\options \
```

# Comandos do scapy

- Se quisermos que o pacote seja enviado para outro IP ou domínio, teremos que configurar a camada IP.
- O comando a seguir criará um pacote nas camadas IP e ICMP:

```
>>> icmp_packet=IP(dst='www.python.org')/ICMP()
```

# Comandos do scapy

- Os métodos **show()** e show2(), nos permitem ver as informações de um pacote específico:

```
>>> icmp_packet.show()
###[ IP ]###
version = 4
ihl               = None
tos               = 0x0
len               = None
id                = 1
flags             =
frag              = 0
ttl               = 64
proto             = icmp
chksum            = None
src               = 192.168.18.21
dst               = Net("www.python.org/32")
\options \
```

```
###[ ICMP ]###
     type        = echo-request
     code        = 0
     chksum      = None
     id          = 0x0
     seq         = 0x0
     unused      = ''
```

# Comandos do scapy

- Os métodos show() e **show2()**, nos permitem ver as informações de um pacote específico:

```
>>> icmp_packet.show2()
###[ IP ]###
version         = 4
ihl             = 5
tos             = 0x0
len             = 28
id              = 1
flags           =
frag            = 0
ttl             = 64
proto           = icmp
chksum          = 0x8bde
src             = 192.168.18.21
dst             = 151.101.132.223
\options \
```

```
###[ ICMP ]###
     type        = echo-request
     code        = 0
     chksum      = 0xf7ff
     id          = 0x0
     seq         = 0x0
     unused      = ''
```

# Comandos do scapy

- Com o seguinte comando, podemos ver a estrutura de um pacote específico:

```
>>> ls(icmp_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                  = <Flag 0 ()>              ('<Flag 0 ()>’)
frag           : BitField (13 bits)          = 0                        ('0’)
ttl            : ByteField                   = 64                       ('64’)
proto          : ByteEnumField               = 1                        ('0’)
chksum         : XShortField                 = None                     ('None’)
src            : SourceIPField               = '192.168.18.21’          ('None’)
dst            : DestIPField                 = Net("www.python.org/32") ('None’)
options        : PacketListField             = []                       ('[]')
```

# Comandos do scapy

- Scapy cria e analisa pacotes camada por camada.
- Os pacotes no scapy são dicionários Python, então cada pacote é um conjunto de dicionários aninhados e cada camada é um dicionário filho da camada principal.
- O método summary() fornece os detalhes das camadas de cada pacote.

```
>>> icmp_packet[0].summary()
'IP / ICMP 192.168.18.21 > Net("www.python.org/32") echo-request 0’
>>> icmp_packet[1].summary()
'ICMP 192.168.18.21 > Net("www.python.org/32") echo-request 0'
```

# Comandos do scapy

- Podemos também criar um pacote sobre outras camadas, como IP/TCP:

```
>>> tcp_packet=IP(dst='python.org')/TCP(dport=80)
>>> tcp_packet.show()
###[ IP ]###
version         = 4
ihl             = None
tos             = 0x0
len             = None
id              = 1
flags           =
frag            = 0
ttl             = 64
proto           = tcp
chksum          = None
src             = 192.168.18.21
dst             = Net("python.org/32")
\options \
###[ TCP ]###
sport           = ftp_data
dport           = www_http
seq             = 0
ack             = 0
dataofs         = None
reserved        = 0
flags           = S
window          = 8192
chksum          = None
urgptr          = 0
options         = ''
```

# Comandos do scapy

- Podemos também criar um pacote sobre outras camadas, como IP/TCP:

```
>>> tcp_packet.show2()
###[ IP ]###
version          = 4
ihl              = 5
tos              = 0x0
len              = 40
id               = 1
flags            =
frag             = 0
ttl              = 64
proto            = tcp
chksum           = 0xdd5b
src              = 192.168.18.21
dst              = 138.197.63.241
\options \
```

```
###[ TCP ]###
     sport       = ftp_data
     dport       = www_http
     seq         = 0
     ack         = 0
     dataofs     = 5
     reserved    = 0
     flags       = S
     window      = 8192
     chksum      = 0xf20a
     urgptr      = 0
     options     = ‘’
>>> tcp_packet.summary()
'IP / TCP 192.168.18.21:ftp_data > Net("python.org/32")
:www_http S'
```

# Enviando Pacotes com scapy

- Para enviar um pacote no scapy, temos disponíveis dois métodos:
  - **send()**: Trabalha com pacotes na camada 3
  - **sendp()**: Trabalha com pacotes na camada 2
- Se precisarmos controlar os pacotes na camada 3 (rede), usamos **send()** para enviar pacotes.
- Se precisarmos controlar os pacotes na camada 2 (Ethernet), usamos **sendp()**.
- Podemos usar o método **help()** nessas duas funções no módulo **scapy.sendrecv** para obter informações de parâmetro.

# Enviando Pacotes com scapy

```
>>> help(send)
send(x, iface=None, **kargs)
        Send packets at layer 3
        :param x: the packets
        :param inter: time (in s) between two packets (default 0)
        :param loop: send packet indefinetly (default 0)
        :param count: number of packets to send (default None=1)
        :param verbose: verbose mode (default None=conf.verbose)
        :param realtime: check that a packet was sent before sending the next one
        :param return_packets: return the sent packets
        :param socket: the socket to use (default is conf.L3socket(kargs))
        :param iface: the interface to send the packets on :
        param monitor: (not on linux) send in monitor mode
        :returns: None
>>> help(sendp)
sendp(x, iface=None, iface_hint=None, socket=None, **kargs)
        Send packets at layer 2
```

# Enviando Pacotes com scapy

- Com o método **send()**, podemos enviar um pacote específico na camada 3 da seguinte forma:

```
>>> send(packet)
```

- Para enviar um pacote da camada 2, podemos usar o método sendp().
- Para usar este método, precisamos adicionar a camada Ethernet e fornecer a interface correta para enviar o pacote:

```
>>> sendp(Ether()/IP(dst="packtpub.com")/ICMP()/"Layer 2 packet",iface="<interface>")
```

- Como já mencionado, esses métodos fornecem alguns parâmetros.
- Por exemplo, com as opções inter e loop, podemos enviar o pacote indefinidamente a cada N segundos:

```
>>> sendp(packet, loop=1, inter=1)
```

# Enviando Pacotes com scapy

- Como o sendp() opera na camada 2, as rotas do sistema não são necessárias, com as informações sendo enviadas diretamente através do adaptador de rede indicado como parâmetro da função.
- Esta função nos permite especificar os endereços MAC do destino.
- Se indicarmos os endereços MAC, o scapy tentará resolvê-los automaticamente com endereços locais e remotos.

# Enviando Pacotes com scapy

- No comando a seguir, geramos um pacote com as camadas Ethernet, IP e ICMP.
- Graças à camada Ether, podemos obter os endereços MAC de origem e destino deste pacote:

```
>>> packet = Ether()/IP(dst="python.org")/ICMP()
>>> packet.show()
###[ Ethernet ]###
    dst       = f4:1d:6b:dd:14:d0
    src       = a4:4e:31:d8:c2:80
    type      = IPv4
```

# Enviando Pacotes com scapy

- Podemos executar essas operações também a partir de um script Python.
- No exemplo a seguir, criamos um pacote ICMP para enviar ao domínio python.org.

```
from scapy.all import *
packet=IP(dst='www.python.org')/ICMP()
packet.show()
sendp(packet)
```

# Enviando Pacotes com scapy

Os métodos send() e sendp() nos permitem enviar as informações que precisamos para a rede, mas não nos permitem receber as respostas.

Existem muitas maneiras de receber respostas dos pacotes que geramos, mas a mais útil é usar os métodos da família sr (derivados da sigla: enviar e receber).

# Enviando Pacotes com scapy

- **sr (...)**: Envia e recebe um pacote, ou uma lista de pacotes para a rede.
  - Espera até que uma resposta seja recebida para todos os pacotes enviados.
  - Esta função funciona na camada 3.
    - Ou seja, para saber como enviar os pacotes, utilize as rotas do sistema.
    - Se não houver rota para enviar o(s) pacote(s) ao destino desejado, ele não poderá ser enviado.
- **sr1 (...)**: Funciona da mesma forma que os métodos sr (...), exceto que captura apenas a primeira resposta recebida e ignora quaisquer outras.
- **srp (...)**: Funciona da mesma forma que o método sr (...), mas na camada 2.
  - Permite enviar informações através de uma interface específica.
  - As informações sempre serão enviadas, mesmo que não haja rota para isso.

# Enviando Pacotes com scapy

- **srp1 (...)**: Seu funcionamento é idêntico ao método sr1 (...), mas funciona na camada 2.
- **srbt (...)**: Envia informações através de uma conexão Bluetooth.
- **srloop (...)**: Permite enviar e receber informações N vezes.
  - Isso significa que podemos, por exemplo, enviar um pacote três vezes e, portanto, receberemos a resposta aos três pacotes, em ordem consecutiva.
  - Também nos permite especificar as ações a serem tomadas quando um pacote é recebido e quando nenhuma resposta é recebida.
- **srploop (...)**: O mesmo que srloop, mas funciona na camada 2.

# Enviando Pacotes com scapy

- Se quisermos enviar e receber pacotes com a possibilidade de ver o pacote de resposta, o método sr1() pode ser útil.
- No exemplo a seguir, construímos um pacote ICMP e o enviamos com sr1():

```
>>> packet=IP(dst='www.python.org')/ICMP()
>>> sr1(packet)
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
<IP version=4 ihl=5 tos=0x0 len=28 id=52517 flags= frag=0 ttl=59 proto=icmp chksum=0xc3b9
src=151.101.132.223 dst=192.168.18.21 |<ICMP type=echo-reply code=0 chksum=0x0 id=0x0
seq=0x0 |>>
```

**Note**:
O pacote é a resposta a uma conexão TCP do domínio Python, onde podemos ver que possui duas camadas (IP e ICMP).

# Enviando Pacotes com scapy

- O script a seguir nos permite conectar com o domínio Python, gerando um pacote com três camadas.

```
from scapy.all import *
packet=Ether()/IP(dst='www.python.org')/TCP(dport=80,flags="S")
packet.show()
srp1(packet, timeout=10)
```

# Enviando Pacotes com scapy

- Outro uso interessante do método srp() junto com as camadas Ether e ARP é obter os hosts ativos em um segmento de rede.
- Por exemplo, para escanear os hosts da nossa sub-rede, bastaria executar o método srp() e exibir os valores dos hosts ativos:

```
>>> answer,unanswer =
srp(Ether(dst="ff:ff:ff:ff:ff")/ARP(pdst="192.168.18.0/24"),timeout=2)
Begin emission: Finished sending 256 packets.
*.*.....................................................
Received 70 packets, got 2 answers, remaining 254 packets
>>> answer.summary()
Ether / ARP who has 192.168.18.1 says 192.168.18.21 ==> Ether / ARP is at
f4:1d:6b:dd:14:d0 says 192.168.18.1
Ether / ARP who has 192.168.18.44 says 192.168.18.21 ==> Ether / ARP is at
e4:75:dc:b3:0e:ec says 192.168.18.44
```

# Continua...