Scapy

Interactive packet management tool. It is a python library and hence we can use direct commands in CLI or we can include in the python program by importing it

This tool can generate, send, sniff, decode and forge any packet

Scapy mainly does two things: sending packets and receiving answers. You define a set of packets, it sends them, receives answers, matches requests with answers and returns a list of packet couples (request, answer) and a list of unmatched packets

Since we can create packets using Scapy we can create packet in the way we want by overloading the default values in packet and send to an intended destination

**Installation of Scapy 2.4.3**

1. git clone <https://github.com/secdev/scapy.git>
2. cd scapy
3. sudo python setup.py install

Scapy can be opened by typing **scapy** with root privilege in the linux terminal

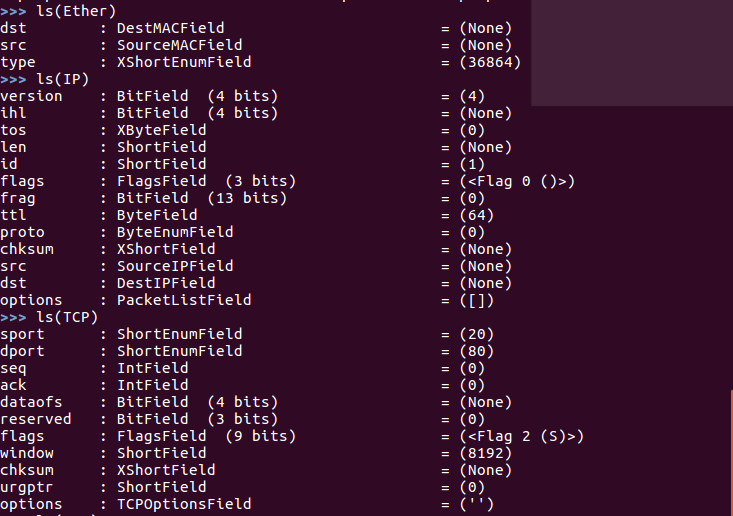
**Basic scapy commands:**

1. ls() - To list all the protocols supported by scapy
2. lsc() - To list all the available commands
3. ls(x) - To list the fields available in each protocol where x = L2 frame, L3 packet,

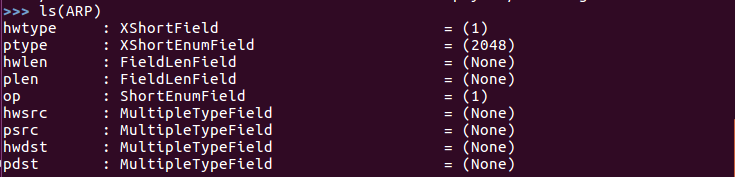
L4 TCP/UDP segments

ls(x)

For Ethernet frame, IP packet and TCP and UDP segments and ARP packet



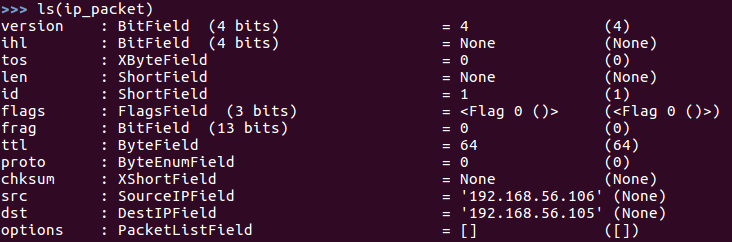




**To create an IP Packet and send it**

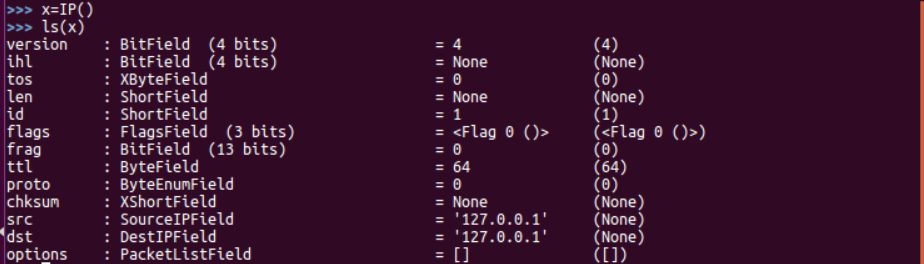


To see how the packet we created look like

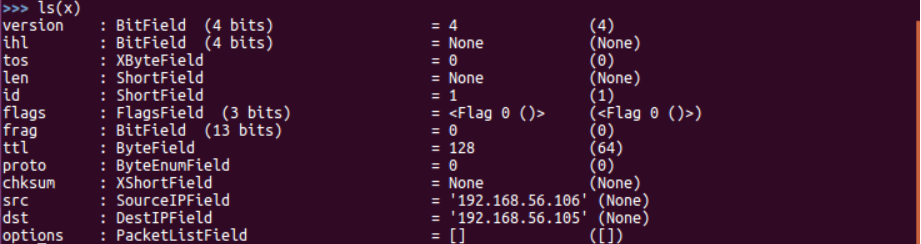


We can also do like below to set the value

Assign the IP() to a variable and then set the value



To set the value of individual fields



The packet x is similar to the packet ip\_packet

We are going to send an icmp echo request using this ip packet



Now we have just sent a packet with the above-mentioned source and destination ip address.

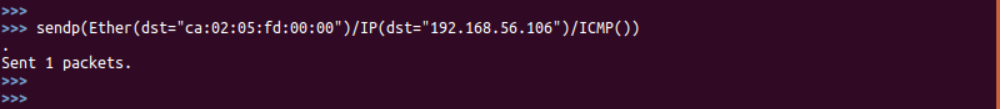
The drawback here is we are not getting any reply . To get a reply and to view the reply we can use the functions sr(), sr1() and srp().These functions can be seen in the upcoming sections.

**To construct packet by changing L2 frame field values and send**

We can also set the value of a fields in L2 frame and send it

Instead of send() we are using sendp() for that purpose since the send() works at L3 and sendp() works at L2

sendp(Ether(src=<src mac address>,dst=<destination mac address>)/ IP(src=<src ip address>,dst=<dst ip address>) /<protocol>)



**Sending and receiving a packet**

To send and receive a packet we have two function

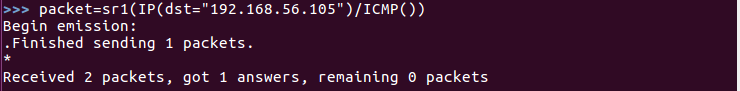
**1. sr1() -** this function sends only one packet and returns answer of the sent packet

2. **sr()** - this function sends many packets and returns the answers sent to those packets

**sr1() –**

For sending a packet

Since the function is returning an answer we are assigning it to a variable in this case packet



To check the answer we received we can use the following function

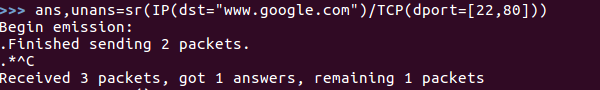


We have got an echo-reply from 192.168.56.105

**sr() –**

For sending more than one packet we go for sr()

Here we are going to send two packets one for ssh request and other for http request to google.com



Here we got answer for one packet and the remaining packet is unanswered

Answered packet



Here we have sent a http request to google.com. The segment has its SYN flag set which can be seen from the **S.** We inturn got a reply from the google server with segment having SYN+ACK bit set which can be seen from **SA.**

Unanswered packet



Here we have sent a ssh request to the google server with the SYN bit set. We have not received any reply from the server. Hence we can see that the server is listening in that port

A layer has default values for every field, so that you don't have to fill them all. If you give a value to the field, it will overload the default value. If you delete the field, the default value will be back**.**

The “/” operator has been used as a composition operator between two layers. When doing so, the lower layer can have one or more of its defaults fields overloaded according to the upper layer.

**How to send a DNS query to a DNS server using scapy**

The command to create and send a DNS query packet is as follows

sr1(IP(dst = <DNS server IP address> ) / UDP() / DNS(rd = 1, qd = DNSQR(qname = <domain name>)))

DNS is a UDP request

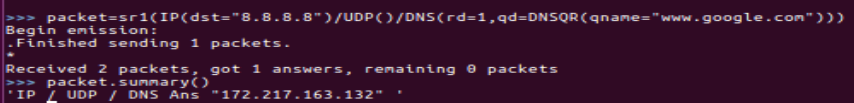
rd - recursion desired. How many times query has to be sent if answer is not obtained from

dns server

qd - query domain

DNSQR - DNS Query Record

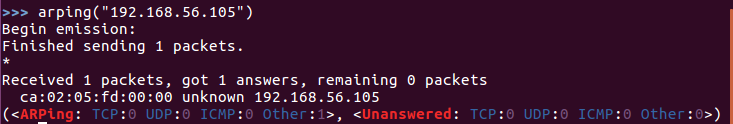
qname - Name of what we wish to query



From the above example, it can be seen that we have obtained the IP address 172.217.163.132 as the ip address of [www.google.com](http://www.google.com) from 8.8.8.8 dns server

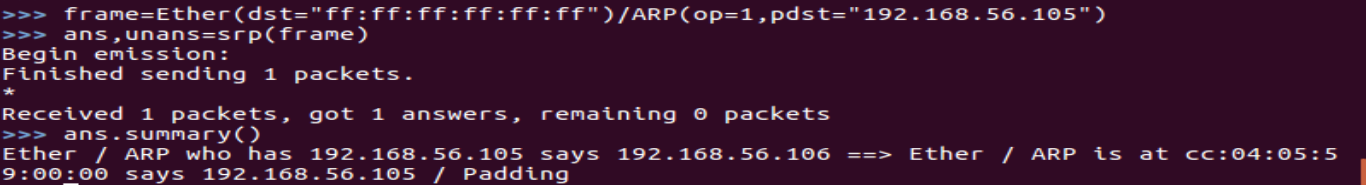
**Sending ARP request to a particular IP or a group of IP**

Using scapy we can send ARP request to a particular IP or a group of IP on the same LAN.



In the above example we have send arp packet to the 192.168.56.105 and we have received a reply with the mac address of the NIC having the IP.

We can also send an ARP broadcast to find out the mac address of the device



**pdst** = packet destination

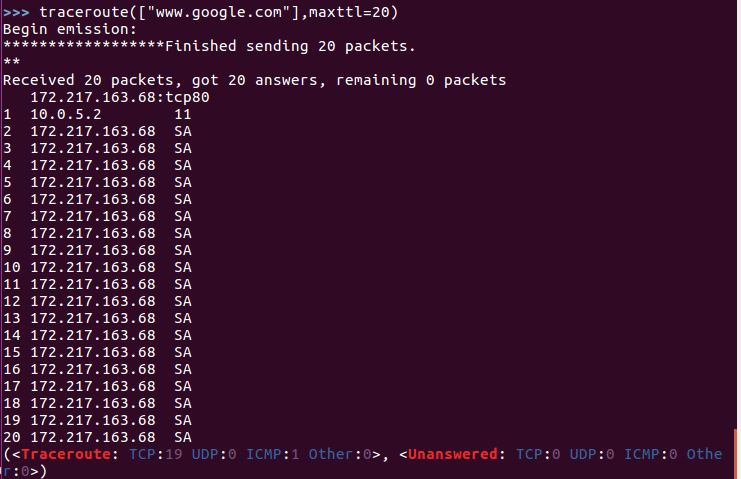
**op**=Operation code op =1 Request ,op =2 Reply

Since we are modifying parameters of layer 2 and sending it we are using srp

Summary of the reply can be seen from the above screenshot

**Traceroute command using Scapy**

Scapy doesn’t use ICMP for traceroute like standard traceroute. Scapy’s traceroute sends all the packets at once and does not wait for individual responses to come hence we use maxttl.



It uses tcp port 80. Scapy uses TCP SYN to port 80 to check if there is a response from the server. If there exists a server it sends TCP ACK for SYN. Hence we are getting SA in the response above.

172.217.163.68 is the IP address of Google LLC at London.

**Sniffing in Scapy**

Scapy uses sniffing to read all the interfaces an scan all the port.

1. To sniff all the ports we can use sniff() with no parameters
2. To sniff a particular interface we can use sniff(iface=”<interface name>”)



The sniff function has sniffed 12016 TCP packets and 20 ICMP packets and 7 other packets. (Scapy sniffer is not super-fast. It can miss packets)

1. To limit the number of packets sniffed we can give the count parameter sniff(count=<number>)



In the above example we have given the count to 20. So only 20 TCP packets are sniffed

1. We can also specify that only particular protocol to be sniffed using filter parameter

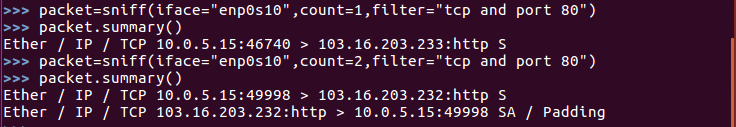
sniff(filter=<Filter condition>)



After giving the command, on Opening any browser a http request will be sent(to generated traffic). Since http packet uses port 80 as destination port and tcp protocol the particular packet is sniffed.

In order to check what has been received we can assign the function to a variable and can use summary function to know what packet has been sniffed.

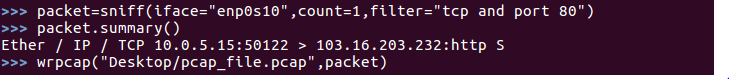
As below:



**To store the packets captured in a pcap file and read a pcap file**

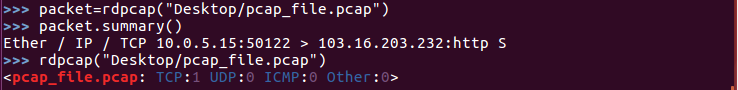
We can store the collected packet in a pcap file by using wrpcap() function

wrpcap(<Destination path>/<filename.pcap>,<packet captured>)



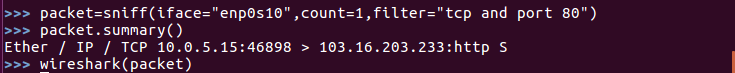
Now the pcap\_file.pcap is saved in the Desktop. The pcap file contains the sniffed packet

To read a pcap file we can use rdpcap(<filename>) command

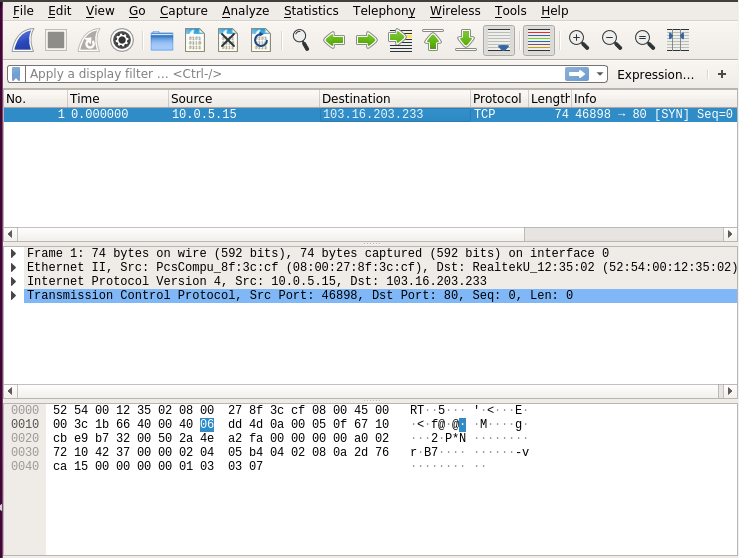


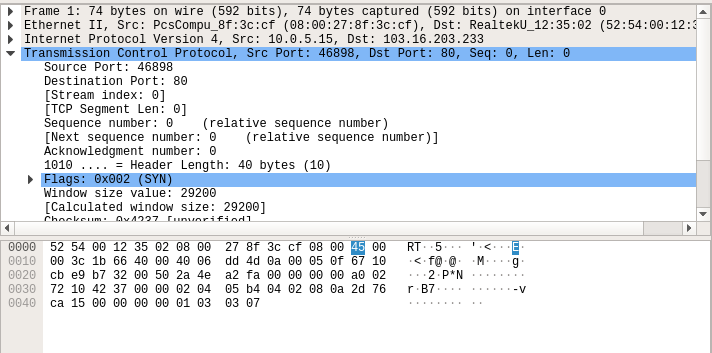
**To open wireshark using scapy and view any sniffed packet**

wireshark(<packet sniffed>)



On giving the above command wireshark opens and the packet/packets sniffed will be shown in the wireshark



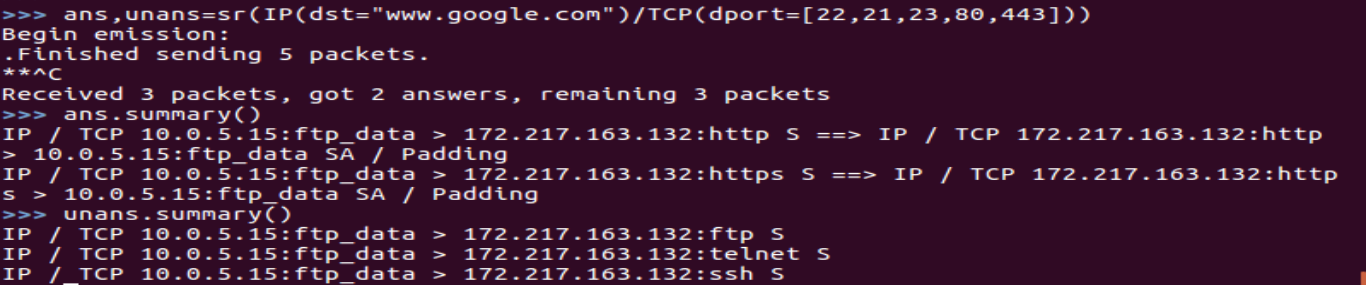


**Some attacks using Scapy**

1. **Port scanning**

An attacker launches a port scan by using a listening service to see what ports are open on the target machine. A port scan attack, therefore, occurs when an attacker sends packets to your machine, which can vary the destination port

Now in the below example we are going to send packet to [www.google.com](http://www.google.com) to port 21,22,23,80 and 443 and we are going to check which of these port are open ports



We have used sr() function to send TCP SYN to all the above mentioned ports

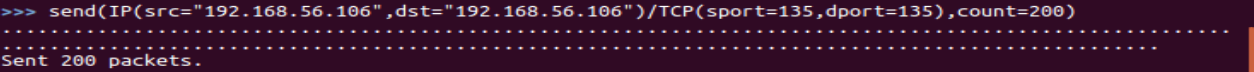
We have SYN ACK from port 80 and 443

We have not received any reply from ports 21,22 and 23

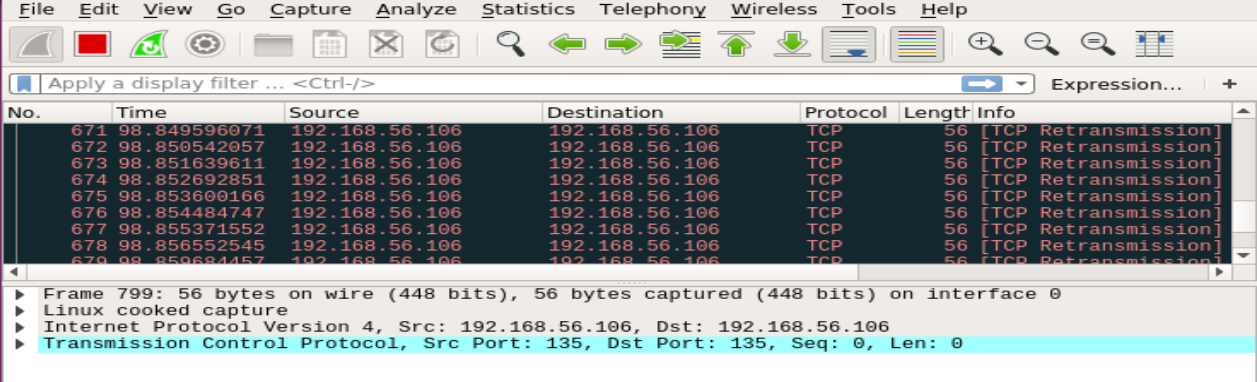
From this we can see that 80 and 443 in google server are open ports and 21,22 and 23 are closed ports

1. **Land attack**

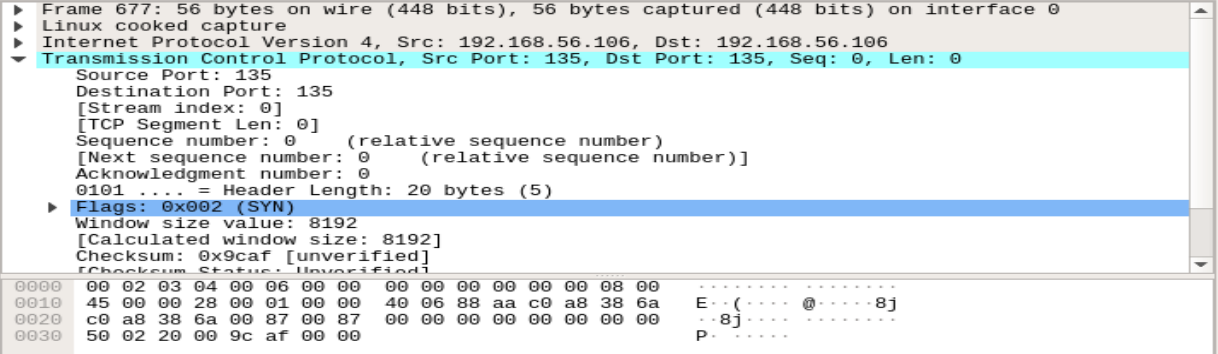
It is a DoS attack that sends an oversized packet to the target with the same source and destination IP address, as well as the same source and destination port. It doesn't always crash the system but will slow it down considerably. For web servers, slowing them down is effectively a DoS



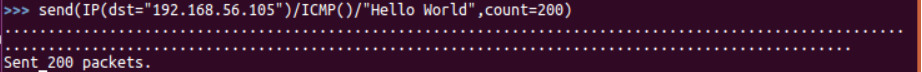
The output captured in wireshark



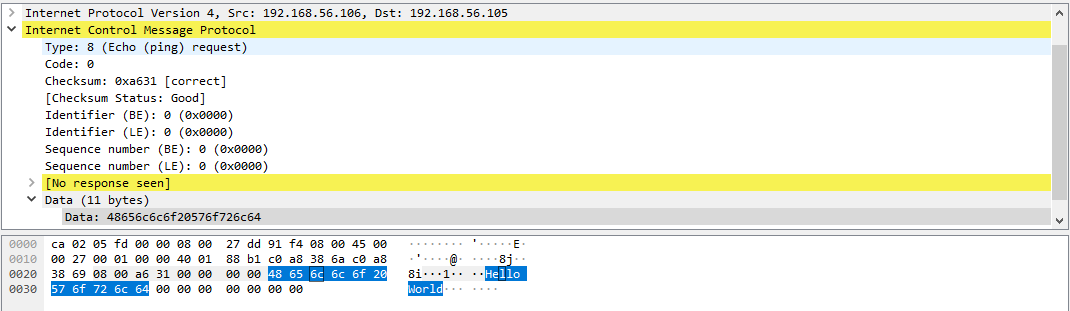
On inspecting a single packet



We can also set the value in the payload and send it using Scapy. As below



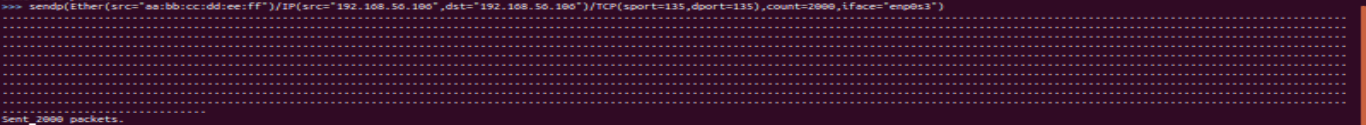
In Wireshark we could see the value in the payload



1. **MAC address spoofing**

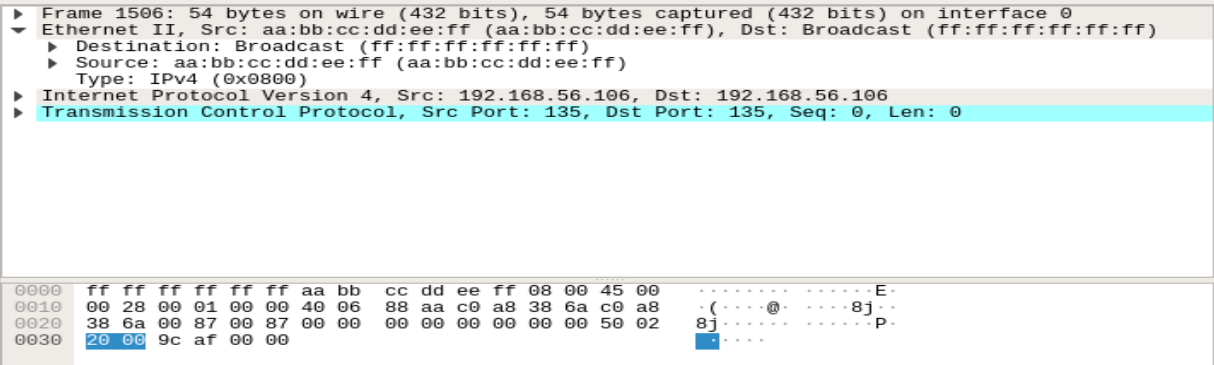
We can change the source mac address by posing a pseudo mac address and send packet using scapy. By this way our real device will be undetectable

sendp(Ether(src=<mac address>,dst=<mac address>) / IP(src=<src ip address>,dst=<dst ip address>) / TCP(sport=<source port>,dport=<destination port>),count=<count of packet to be sent,iface=<device interface through which the packet must be sent>)



Here we are posing as mac address aa:bb:cc:dd:ee:ff and we are sending the frames

While capturing in wireshark we can see that the packet is captured with the encapsulated frame having the pseudo mac address



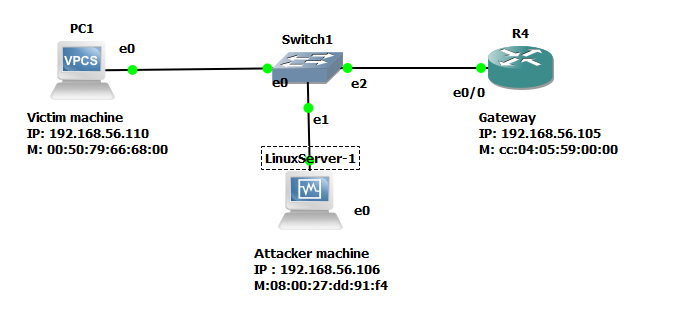
We have successfully spoofed the mac address of the source device using scapy

1. **ARP Cache Poisoning**

Machine A (192.168.56.110) wants to communicate to MachineB (192.168.56.105). In order to communicate, Machine A requires the MAC address of MachineB. So, Machine A searches its ARP table(cache) if it could find the MAC address associated with the IP address 192.168.56.105. If it does, well and good it can send the packet to Machine B, else Machine A will send an ARP broadcast message. The ARP broadcast is directed to ff:ff:ff:ff:ff:ff. The request message will travel across the network to every machine asking if that machine’s IP address matches 192.168.56.105. When machine B finds the ARP request, it sends an ARP response to Machine A(192.168.56.110) telling what its MAC address is. Machine A writes this to its local ARP table.

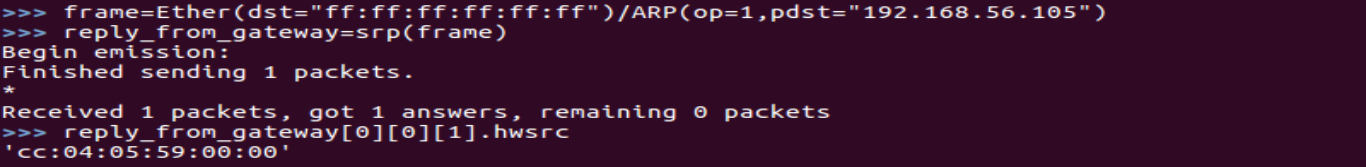
To poison it, we will send false ARP responses, by **spoofing the source IP address**.

The set up we are going to use is in the below diagram

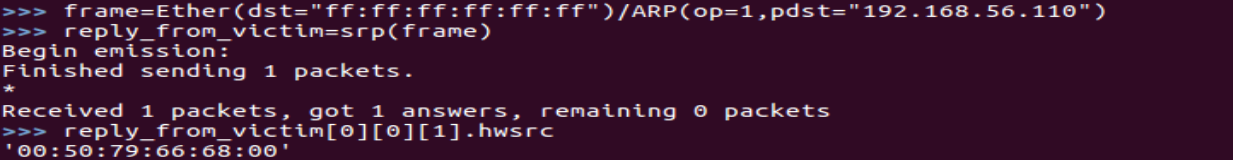


From the attacker machine we can find the mac address of the Victim machine and Gateway by using ARP broadcast

Mac address of the gateway



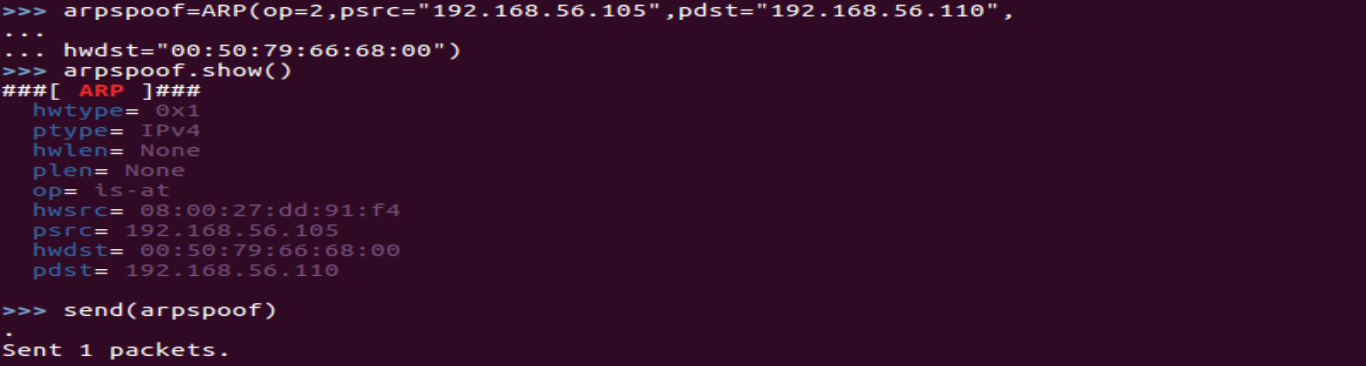
Mac address of the victim machine



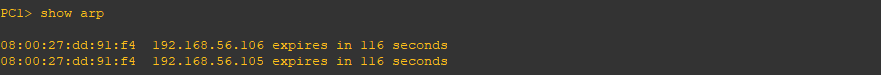
We are basically getting the hardware address of the machine that sent the reply for the ARP request

Now we are going to send an ARP packet to the victim machine with the following parameters

1. HW source address = attacker machine address
2. Source IP address = router gateway IP
3. HW destination address = victim machine address
4. Destination IP address = victim IP



Since in this ARP packet the hardware source address of the attacker is mapped to the IP address of the gateway, the same will be stored in cache of the victim

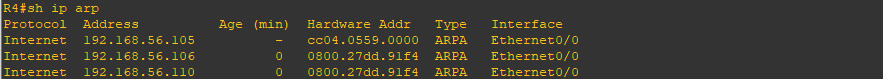


The same has to be done at the gateway side to establish full arp poisoning attack

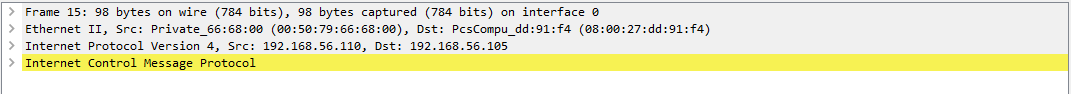
1. HW source address = attacker machine address
2. Source IP address = victim IP
3. HW destination address = router gateway machine address
4. Destination IP address = router gateway IP



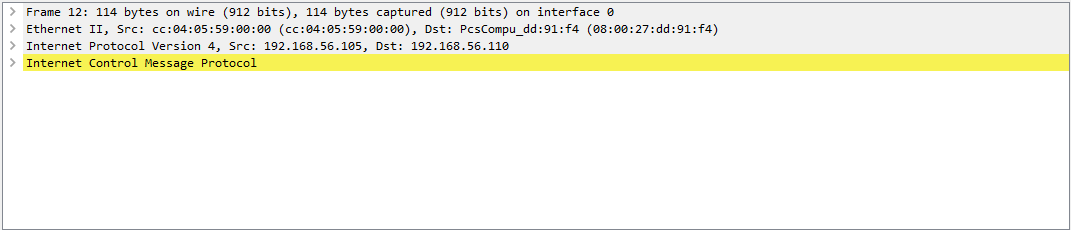
Since in this ARP packet the hardware source address of the attacker is mapped to the IP address of the victim, the same will be stored in cache of the router



Ping from victim computer to router gateway as captured by wireshark



Ping from the gateway to the victim computer as captured by wireshark



Since the arp cache expires within some time the false arppacket has to be sent to the router as well as to the victim machine in a cycle