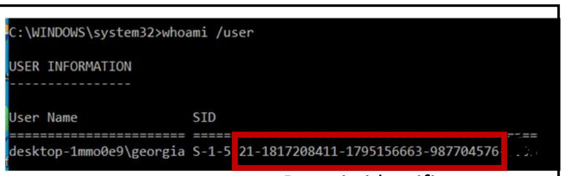
# Enumerating Users

## Chcking on SMB port

- SMB: server Message Block
  - protocol for sharing resources like files, printers, in general any resource which should be retrievable or made available by the server
- SMB port: 445 or 139
  - Default service in Windows OS
  - Non-default service in Linux OS (samba server needs to be installed)
- Security flaws
  - No strong password or Default settings
  - Samba server vulnerability
- Let’s check…
  - Nmap –sC –p139, 445 192.168.84.181

# Enumerating SIDs



```
C:\WINDOWS\system32>whoami /user

USER INFORMATION
----------------

User Name              SID
====================== ====
desktop-1mmo0e9\georgia S-1-5 21-1817208411-1795156663-987704576-
```

Domain identifier

- On Windows, each group and account have a unique security identifier (SID)
  - ❖C:\> whoami /user (*Run as administrator)
- SID: Consist of S-[X]-[Y]-[domain/computer]-RID
  - ❖X is the revision level (typically 1)
  - ❖Y is an authority level (typically 5 for user and group)

(*SID: object's security identifier for user, process, group, etc.)

- RID: relative ID, a unique number for the given account or group
  - ❖Original administrator account has RID 500
  - ❖Guest account has RID 501
  - ❖User created on the machine have RIDs 1000 and up

(*RID: incremental portion of SID)

# Enumerating Users using User2sid and Sid2user

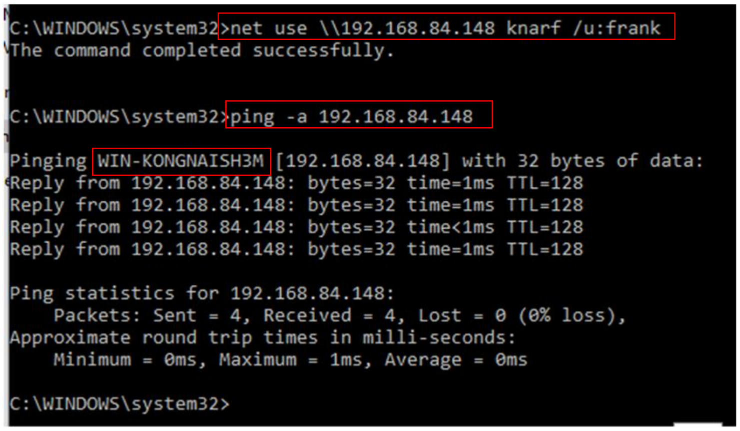
- 1) Get SID from user data 2) Then find potential users using RID
- Start by establishing an SMB session (assuming port 445 is open)

❖C:\> **net use** \\[targetIP] [password] /u:[user]

  - ❖ mapping network drives to your local computer

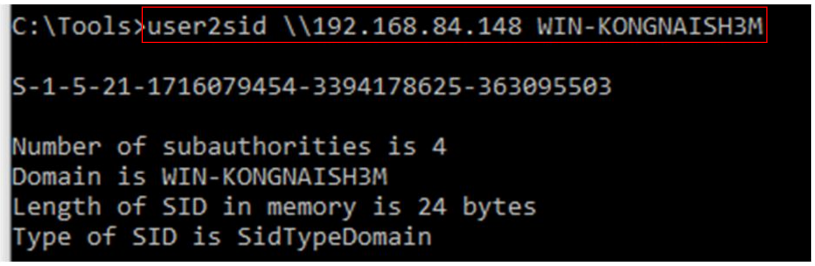
- Obtain domain/computer component of the SID
  - ❖C:\> **user2sid** \\[targetIP] [hostname]
  - ❖You could get hostname from **ping –a** command
- Lookup potential users based on their RIDS
  - ❖C:\> **for /L %i in (1000,1,1020) do @sid2user \\[targetIP] [SID without RID] %i**

# Setting Up the SMB Session and Finding the Hostname



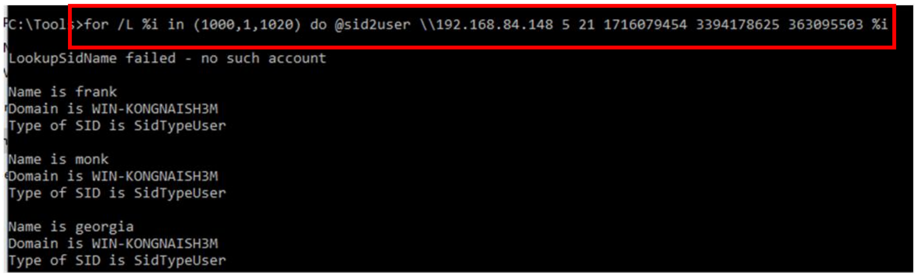
```
C:\WINDOWS\system32>net use \\192.168.84.148 knarf /u:frank
The command completed successfully.


C:\WINDOWS\system32>ping -a 192.168.84.148

Pinging WIN-KONGNAISH3M [192.168.84.148] with 32 bytes of data:
Reply from 192.168.84.148: bytes=32 time=1ms TTL=128
Reply from 192.168.84.148: bytes=32 time=1ms TTL=128
Reply from 192.168.84.148: bytes=32 time<1ms TTL=128
Reply from 192.168.84.148: bytes=32 time=1ms TTL=128

Ping statistics for 192.168.84.148:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\WINDOWS\system32>
```

# Obtaining SID Using user2sid

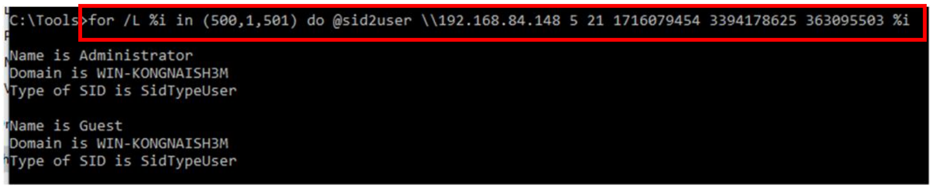


```
C:\Tools>user2sid \\192.168.84.148 WIN-KONGNAISH3M

S-1-5-21-1716079454-3394178625-363095503

Number of subauthorities is 4
Domain is WIN-KONGNAISH3M
Length of SID in memory is 24 bytes
Type of SID is SidTypeDomain
```

# Enumerating Users Using sid2user



```
C:\Tools>for /L %i in (1000,1,1020) do @sid2user \\192.168.84.148 5 21 1716079454 3394178625 363095503 %i
LookupSidName failed - no such account

Name is frank
Domain is WIN-KONGNAISH3M
Type of SID is SidTypeUser

Name is monk
Domain is WIN-KONGNAISH3M
Type of SID is SidTypeUser

Name is georgia
Domain is WIN-KONGNAISH3M
Type of SID is SidTypeUser
```



```
C:\Tools>for /L %i in (500,1,501) do @sid2user \\192.168.84.148 5 21 1716079454 3394178625 363095503 %i

Name is Administrator
Domain is WIN-KONGNAISH3M
Type of SID is SidTypeUser

Name is Guest
Domain is WIN-KONGNAISH3M
Type of SID is SidTypeUser
```

# Enumerating Using rpcclient from Linux

- If you discover a server running the SMB protocol you can test if it’s vulnerable to anonymous connection (also called **null session**) and then glean a lot of informations with a RPC client.

- Via the SAMBA project, Linux also has SMB implementations including SMB client tools such as smbclient and rpcclient

- To establish a SMB session with a Windows box using rpcclient, run
  - ❖$ rpcclient –U username Win_IP_Address

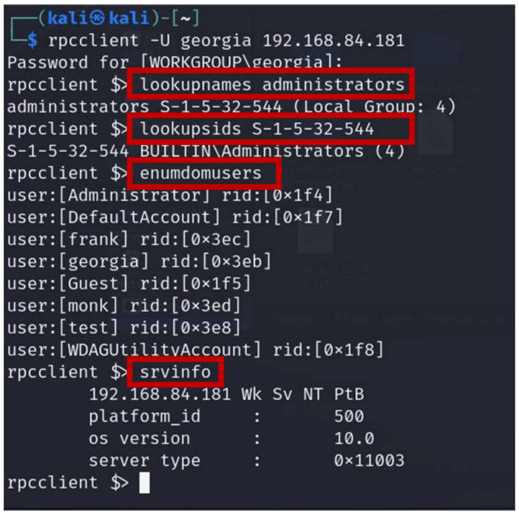
- After providing a password, you will recieve the rpcclient prompt
  - ❖rpcclient $>

# Rpcclient Commands

- **help:** get help
- **enumdomusers:** list users defined locally on the machine, as well as any domain users the system knows about
- **enumalsgroups [domain] | [builtin]:** list groups
- **lsaenumsid:** show all users' sids defined locally on the target Windows box
- **lookupnames [name]:** show SID associated with user or group name
- **lookupsids [sid]:** show user name associated with SID
- **srvinfo:** show OS type and version

# Enumerating Admin Group Membership and Server Info

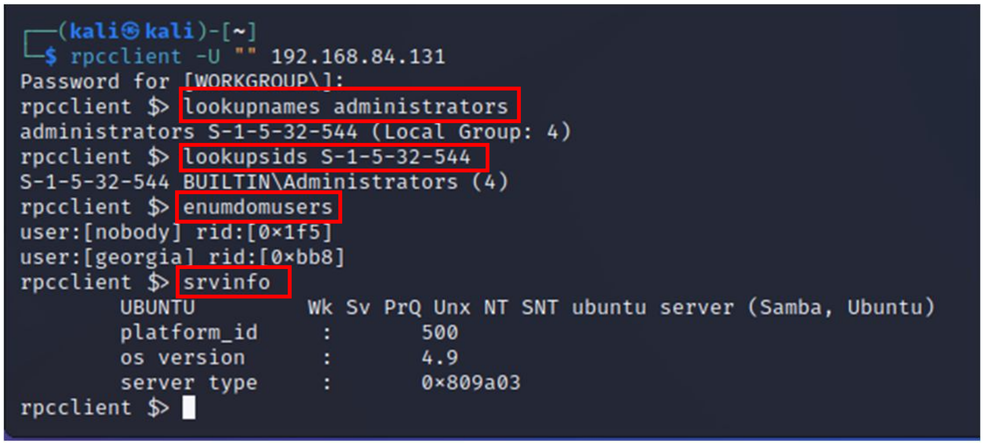
- Tried Windows 10 (from Linux to Windows)



```
┌──(kali㉿kali)-[~]
└─$ rpcclient -U georgia 192.168.84.181
Password for [WORKGROUP\georgia]:
rpcclient $> lookupnames administrators
administrators S-1-5-32-544 (Local Group: 4)
rpcclient $> lookupsids S-1-5-32-544
S-1-5-32-544 BUILTIN\Administrators (4)
rpcclient $> enumdomusers
user:[Administrator] rid:[0×1f4]
user:[DefaultAccount] rid:[0×1f7]
user:[frank] rid:[0×3ec]
user:[georgia] rid:[0×3eb]
user:[Guest] rid:[0×1f5]
user:[monk] rid:[0×3ed]
user:[test] rid:[0×3e8]
user:[WDAGUtilityAccount] rid:[0×1f8]
rpcclient $> srvinfo
        192.168.84.181 Wk Sv NT PtB
        platform_id     :       500
        os version      :       10.0
        server type     :       0×11003
rpcclient $> █
```

## Enumerating Admin Group Membership and Server Info

- Tried null session (Ubuntu), worked

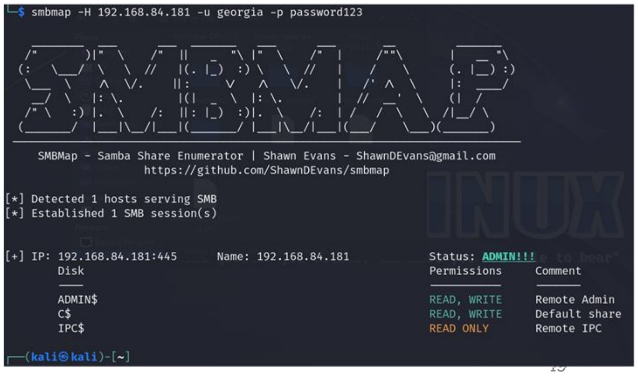


```
┌──(kali㉿kali)-[~]
└─$ rpcclient -U "" 192.168.84.131
Password for [WORKGROUP\]:
rpcclient $> lookupnames administrators
administrators S-1-5-32-544 (Local Group: 4)
rpcclient $> lookupsids S-1-5-32-544
S-1-5-32-544 BUILTIN\Administrators (4)
rpcclient $> enumdomusers
user:[nobody] rid:[0×1f5]
user:[georgia] rid:[0×bb8]
rpcclient $> srvinfo
        UBUNTU         Wk Sv PrQ Unx NT SNT ubuntu server (Samba, Ubuntu)
        platform_id     :       500
        os version      :       4.9
        server type     :       0×809a03
rpcclient $> █
```

## SMBMap

- SMBMap allows users to enumerate samba share drives across an entire domain. List share drives, drive permissions, share contents, upload/download functionality, file name auto-download pattern matching, and even execute remote commands

$ smbmap -H 192.168.84.181 -u georgia -p password123



```
└─$ smbmap -H 192.168.84.181 -u georgia -p password123

    ________  ___      ___  _______   ___      ___       __         _______
   /"       )|"  \    /"  ||   _  "\ |"  \    /"  |     /""\       |   __ "\
  (:   \___/  \   \  //   |(. |_)  :) \   \  //   |    /    \      (. |__) :)
   \___  \    /\  \/.    ||:     \/   /\   \/.    |   /' /\  \     |:  ____/
    __/  \   |: \.        |(|  _  \  |: \.        |  //  __'  \    (|  /
   /" \   :) |.  \    /:  ||: |_)  :)|.  \    /:  | /   /  \   \  /|__/ \
  (_______/  |___|\__/|___|(_______/ |___|\__/|___|(___/    \___)(_______)
 -----------------------------------------------------------------------------
     SMBMap - Samba Share Enumerator | Shawn Evans - ShawnDEvans@gmail.com
                     https://github.com/ShawnDEvans/smbmap

[*] Detected 1 hosts serving SMB
[*] Established 1 SMB session(s)

[+] IP: 192.168.84.181:445      Name: 192.168.84.181            Status: ADMIN!!!
        Disk                                                    Permissions     Comment
        ----                                                    -----------     -------
        ADMIN$                                                  READ, WRITE     Remote Admin
        C$                                                      READ, WRITE     Default share
        IPC$                                                    READ ONLY       Remote IPC

┌──(kali㉿kali)-[~]
```

# Chapter 07
# Capturing Traffic

17

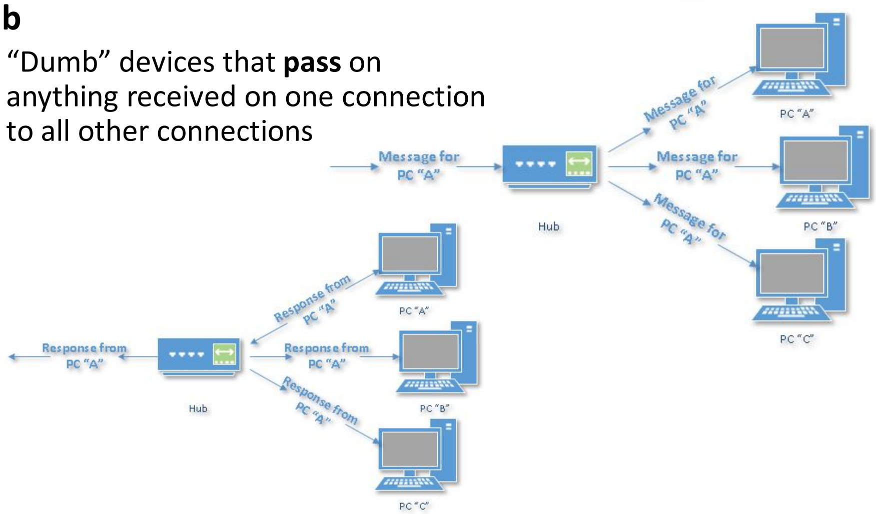
## Outline

- Networking for Capturing Traffic
- Using Tcpdump
- Scapy
- Using Wireshark
- ARP Cache Poisoning
- DNS Cache Poisoning

18

# Hub, Switch, & Router

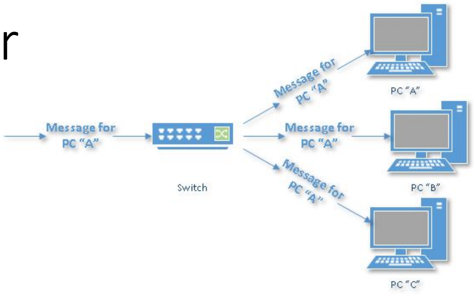
- **Hub**
  - “Dumb” devices that **pass** on anything received on one connection to all other connections

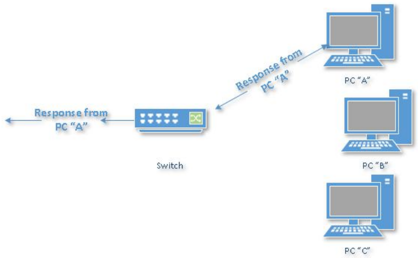


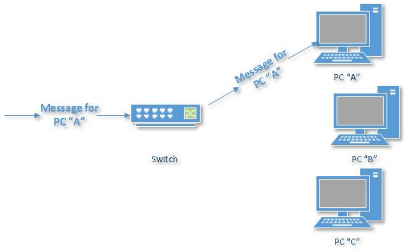
19

# Hub, Switch, & Router

- **Switch**
  - Switches are semi-intelligent devices that **learn** which devices are on which connection



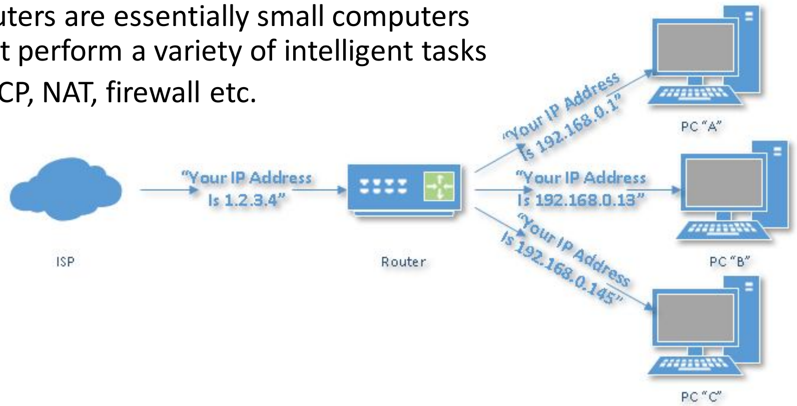




20

# Hub, Switch, & Router

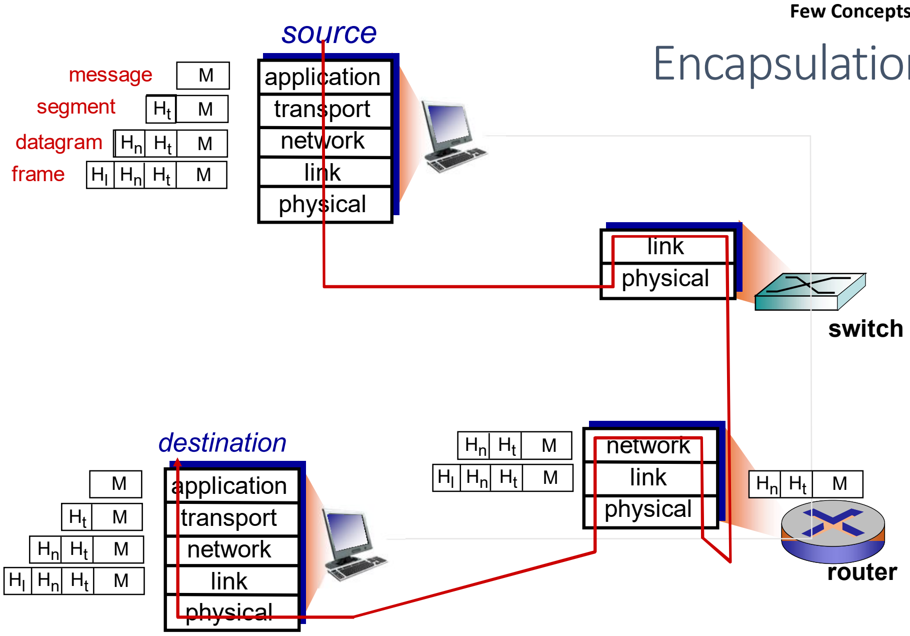
- **Router**
  - Routers are essentially small computers that perform a variety of intelligent tasks
  - DHCP, NAT, firewall etc.



21

Few Concepts

## Encapsulation

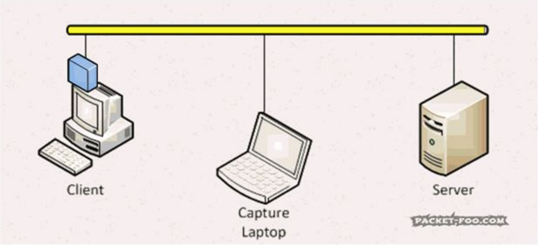


22

# Tcpdump

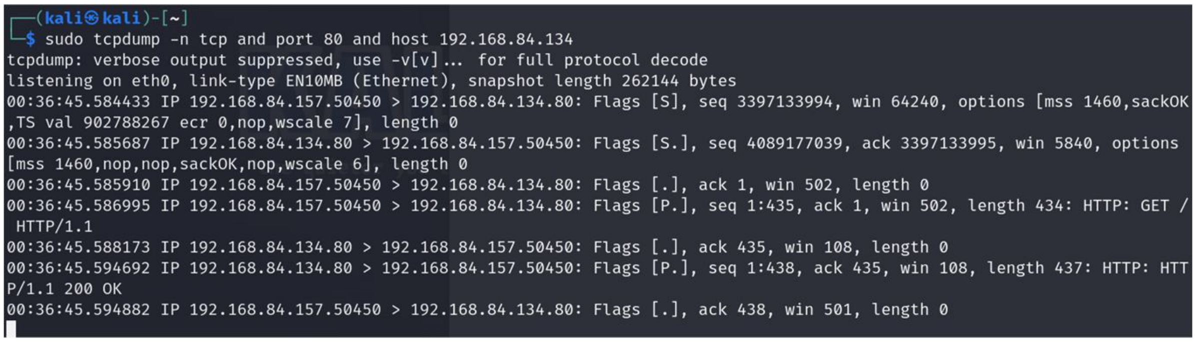
TCPDUMP

- Free open source **sniffer**, Command-line **packet analyzer**. Ported to Windows as WinDump
- The tool should be invoked with root-level privileges to make sure it can put the interface into promiscuous mode (grabbing all packets that pass by the network interface)

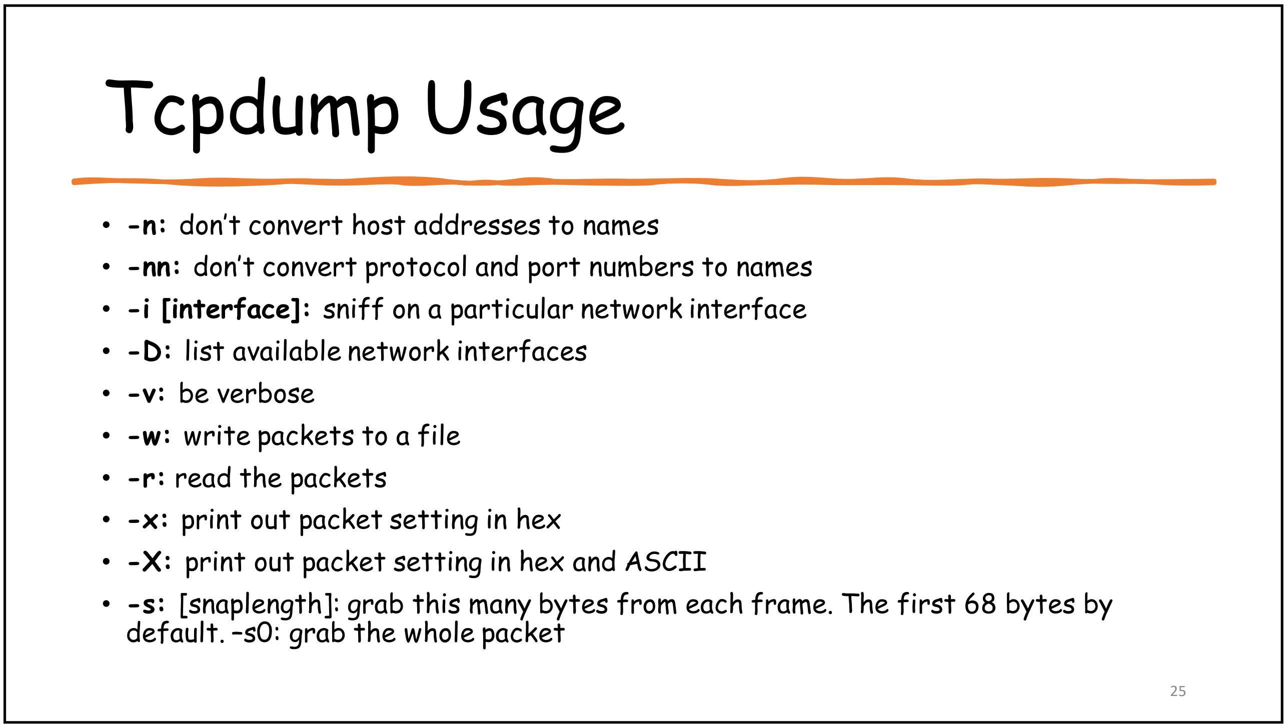


# Example

- Show all TCP port 443 packets going to or from host 192.168.84.181
  - ❖# sudo tcpdump -n tcp and port 80 and host 192.168.84.181

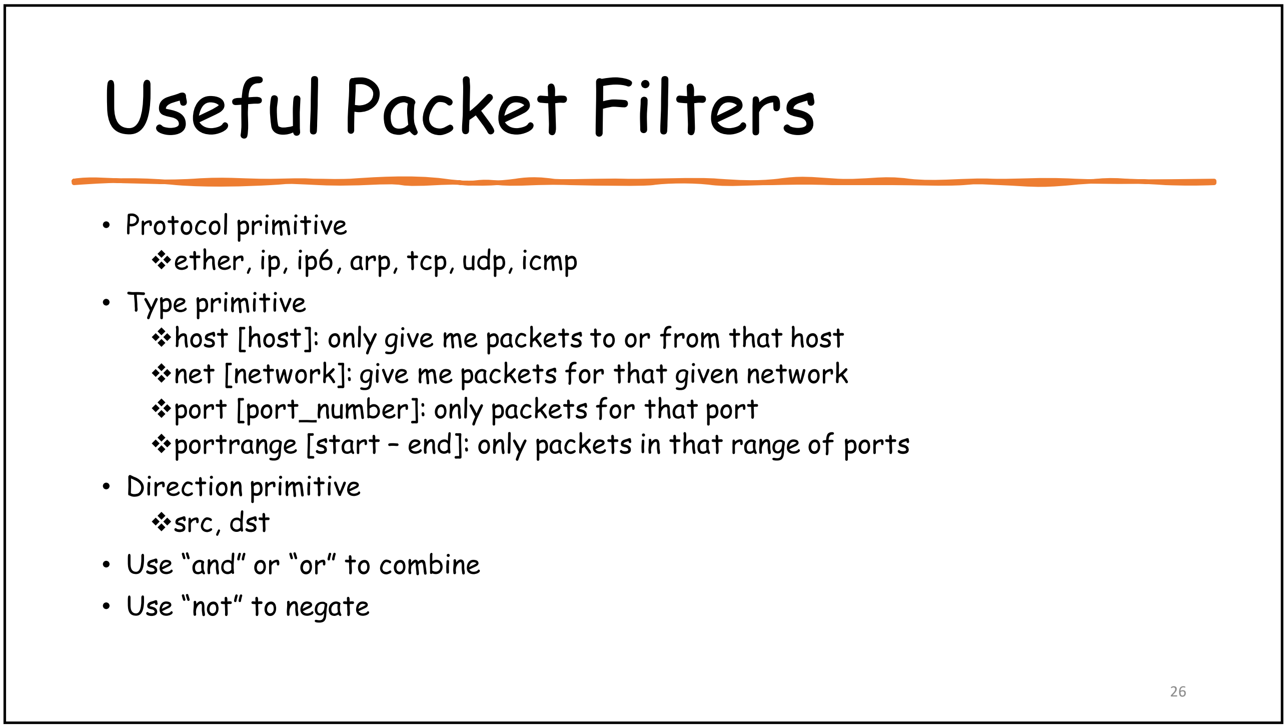


```
┌──(kali㉿kali)-[~]
└─$ sudo tcpdump -n tcp and port 80 and host 192.168.84.134
tcpdump: verbose output suppressed, use -v[v]...  for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
00:36:45.584433 IP 192.168.84.157.50450 > 192.168.84.134.80: Flags [S], seq 3397133994, win 64240, options [mss 1460,sackOK
,TS val 902788267 ecr 0,nop,wscale 7], length 0
00:36:45.585687 IP 192.168.84.134.80 > 192.168.84.157.50450: Flags [S.], seq 4089177039, ack 3397133995, win 5840, options
[mss 1460,nop,nop,sackOK,nop,wscale 6], length 0
00:36:45.585910 IP 192.168.84.157.50450 > 192.168.84.134.80: Flags [.], ack 1, win 502, length 0
00:36:45.586995 IP 192.168.84.157.50450 > 192.168.84.134.80: Flags [P.], seq 1:435, ack 1, win 502, length 434: HTTP: GET /
 HTTP/1.1
00:36:45.588173 IP 192.168.84.134.80 > 192.168.84.157.50450: Flags [.], ack 435, win 108, length 0
00:36:45.594692 IP 192.168.84.134.80 > 192.168.84.157.50450: Flags [P.], seq 1:438, ack 435, win 108, length 437: HTTP: HTT
P/1.1 200 OK
00:36:45.594882 IP 192.168.84.157.50450 > 192.168.84.134.80: Flags [.], ack 438, win 501, length 0
```



# Tcpdump Usage

- **-n:** don't convert host addresses to names
- **-nn:** don't convert protocol and port numbers to names
- **-i [interface]:** sniff on a particular network interface
- **-D:** list available network interfaces
- **-v:** be verbose
- **-w:** write packets to a file
- **-r:** read the packets
- **-x:** print out packet setting in hex
- **-X:** print out packet setting in hex and ASCII
- **-s:** [snaplength]: grab this many bytes from each frame. The first 68 bytes by default. -s0: grab the whole packet



# Useful Packet Filters

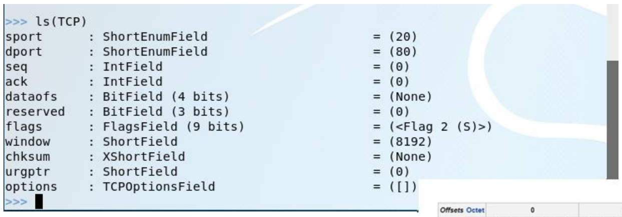
- Protocol primitive
  - ❖ether, ip, ip6, arp, tcp, udp, icmp
- Type primitive
  - ❖host [host]: only give me packets to or from that host
  - ❖net [network]: give me packets for that given network
  - ❖port [port_number]: only packets for that port
  - ❖portrange [start – end]: only packets in that range of ports
- Direction primitive
  - ❖src, dst
- Use "and" or "or" to combine
- Use "not" to negate

# Scapy

- Scapy is a **packet crafting, manipulation and analysis suite**
- Scapy is an environment based on **Python**
- To craft packets with Scapy, you have to **invoke it with UID 0 privileges** on Linux
  - ❖# scapy
- Exit Scapy, press CTRL-D

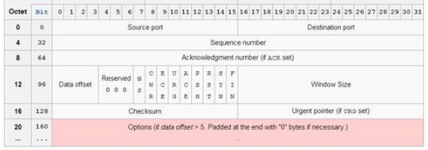
# Listing Protocols

- Use ls() command to list all protocols supported by Scapy
- ARP, IP, ICMP, TCP, UDP, Ether, etc.
- To see the fields you can set within a given protocol, type ls(*[PROTO]*)



```
>>> 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 (9 bits)                = (<Flag 2 (S)>)
window     : ShortField                         = (8192)
chksum     : XShortField                        = (None)
urgptr     : ShortField                         = (0)
options    : TCPOptionsField                    = ([])
>>> 
```

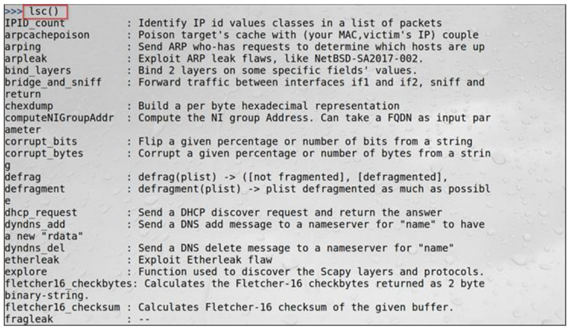
TCP Header



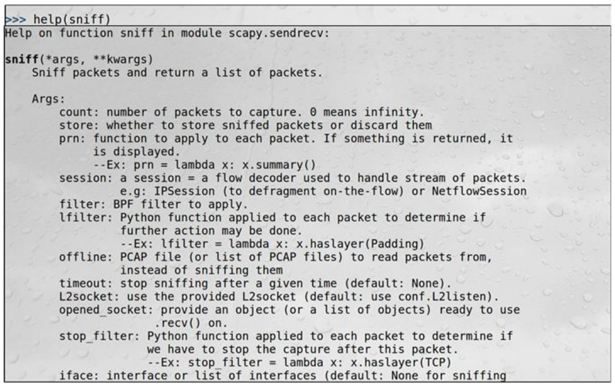
<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td>4</td><td>32</td><td colspan="32">Sequence number</td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4">Data offset</td><td colspan="3">Reserved<br>0 0 0</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Window Size</td></tr>
<tr><td>16</td><td>128</td><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><td>20<br>…</td><td>160<br>...</td><td colspan="32">Options (if <i>data offset</i> &gt; 5. Padded at the end with "0" bytes if necessary.)<br>…</td></tr>
</table>

# Listing Commands

- lsc() commands shows all Scapy functions
- To get help with any function, type
  - ❖help([function])

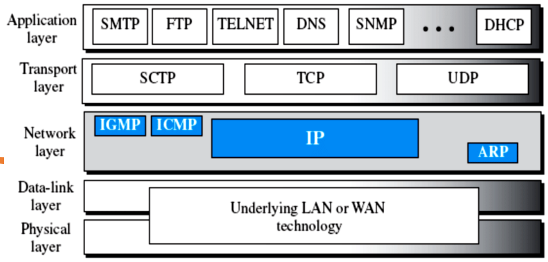


```
>>> lsc()
IPID_count          : Identify IP id values classes in a list of packets
arpcachepoison      : Poison target's cache with (your MAC,victim's IP) couple
arping              : Send ARP who-has requests to determine which hosts are up
arpleak             : Exploit ARP leak flaws, like NetBSD-SA2017-002.
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 input par
ameter
corrupt_bits        : Flip a given percentage or number of bits from a string
corrupt_bytes       : Corrupt a given percentage or number of bytes from a strin
g
defrag              : defrag(plist) -> ([not fragmented], [defragmented],
defragment          : defragment(plist) -> plist defragmented as much as possibl
e
dhcp_request        : Send a DHCP discover request and return the answer
dyndns_add          : Send a DNS add message to a nameserver for "name" to have
a new "rdata"
dyndns_del          : Send a DNS delete message to a nameserver for "name"
etherleak           : Exploit Etherleak flaw
explore             : Function used to discover the Scapy layers and protocols.
fletcher16_checkbytes: Calculates the Fletcher-16 checkbytes returned as 2 byte
binary-string.
fletcher16_checksum : Calculates Fletcher-16 checksum of the given buffer.
fragleak            : --
```



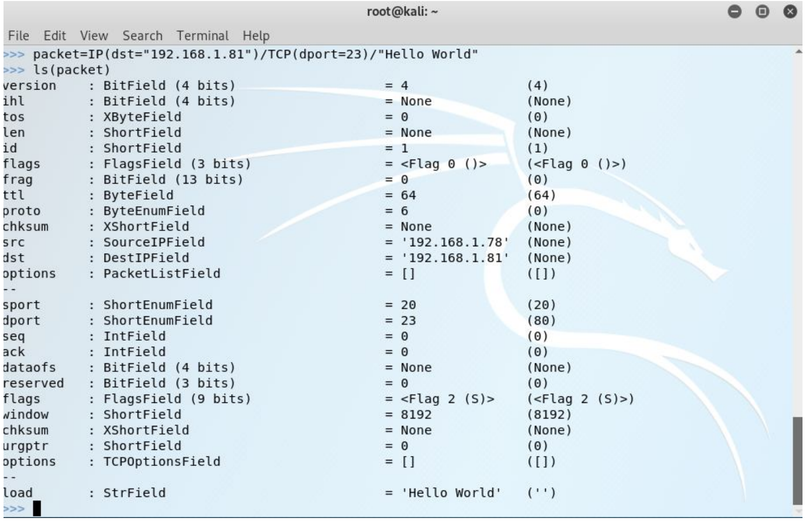
```
>>> help(sniff)
Help on function sniff in module scapy.sendrecv:

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.
                 e.g: IPSession (to defragment on-the-flow) or NetflowSession
        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
        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)
        iface: interface or list of interfaces (default: None for sniffing
```

# Making Packets



- Packets are constructed by layers
- Build from lower layers up to higher layers moving **from left to right**
- Separate layers with **/**
- Override default value for field with <field>=<value>
  - ❖packet=IP(dst="192.168.1.81")/TCP(dport=23)/"Hello World"
  - ❖ls(packet)

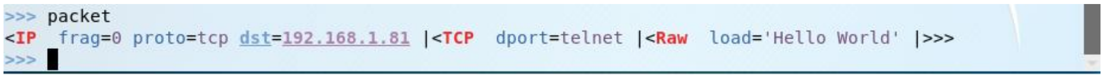
# Making Packets



```
>>> packet=IP(dst="192.168.1.81")/TCP(dport=23)/"Hello World"
>>> ls(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 (3 bits)                   = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)                    = 0               (0)
ttl        : ByteField                             = 64              (64)
proto      : ByteEnumField                         = 6               (0)
chksum     : XShortField                           = None            (None)
src        : SourceIPField                         = '192.168.1.78'  (None)
dst        : DestIPField                           = '192.168.1.81'  (None)
options    : PacketListField                       = []              ([])
--
sport      : ShortEnumField                        = 20              (20)
dport      : ShortEnumField                        = 23              (80)
seq        : IntField                              = 0               (0)
ack        : IntField                              = 0               (0)
dataofs    : BitField (4 bits)                     = None            (None)
reserved   : BitField (3 bits)                     = 0               (0)
flags      : FlagsField (9 bits)                   = <Flag 2 (S)>    (<Flag 2 (S)>)
window     : ShortField                            = 8192            (8192)
chksum     : XShortField                           = None            (None)
urgptr     : ShortField                            = 0               (0)
options    : TCPOptionsField                       = []              ([])
--
load       : StrField                              = 'Hello World'   ('')
>>> 
```

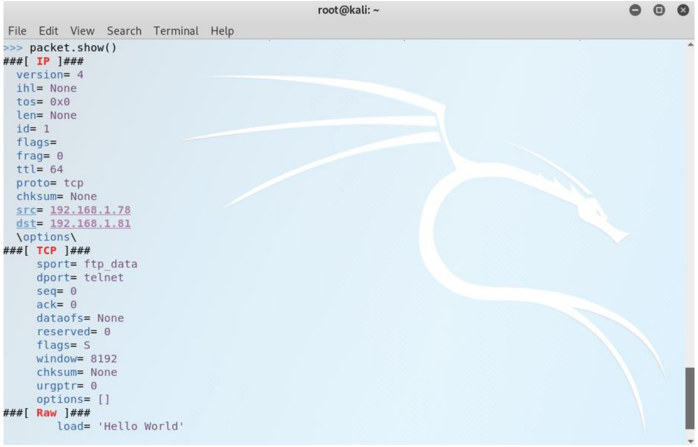
# Inspecting Packet commands

- packet
- packet.summary()
- packet.show()
- ls(packet)



```
>>> packet
<IP  frag=0 proto=tcp dst=192.168.1.81 |<TCP  dport=telnet |<Raw  load='Hello World' |>>>
>>> 
```

# Inspecting Packets



```
>>> packet.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 192.168.1.78
  dst= 192.168.1.81
  \options\
###[ TCP ]###
     sport= ftp_data
     dport= telnet
     seq= 0
     ack= 0
     dataofs= None
     reserved= 0
     flags= S
     window= 8192
     chksum= None
     urgptr= 0
     options= []
###[ Raw ]###
        load= 'Hello World'
```

# Inspecting Packet Fields

- You can view the value of an individual field in a packet by using packet_name.field_name (ex. packet.dport)
- If the field name is not unique across the protocol layers, use packet_name[proto].field_name
- For example
  - ❖**packet[TCP].flags**
- After creating a packet, you can change any field in the packet by using packet_name.field_name = value
- If the field is not unique, use packet_name[proto].field_name = value

# Specifying Addresses and Ports

- Single target
  - ❖Packet=IP(dst="198.162.1.81")
- CIDR notation
  - ❖Packet=IP(dst="198.162.1.**0/24**")
- **Multiple targets**
  - ❖Packet=IP(dst=["198.162.1.1","198.162.1.3","198.162.1.6"])
- Create packet destined for **ports 1-1000**
  - ❖Packet=IP(dst="198.162.1.81")/TCP(dport=(1,1000))
- For **a list of ports**, type
  - ❖Packet=IP(dst="198.162.1.81")/TCP(dport=[23,80,443])

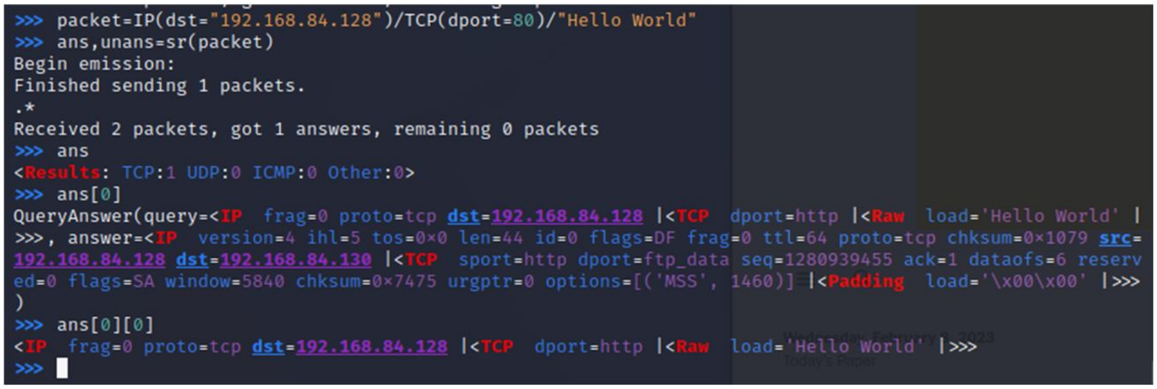
# Sending Packets

- **send():** send packets and don't receive any response back
- **sr():** send and receive packets
- **sr1():** send packets and returns only the first answer
- **srloop():** send the same packets continuously

- Most of the send/receive functions have the following options
  - ❖filter=[bpf packet filter]
  - ❖count=N: send N packets
  - ❖retry=N: resend packet up to N times if no response is received
  - ❖timeout=N: wait only N seconds for a response

# Inspecting Results

- Response from Metasploitable http server

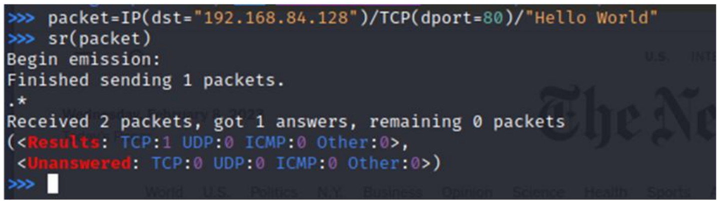


```
>>> packet=IP(dst="192.168.84.128")/TCP(dport=80)/"Hello World"
>>> ans,unans=sr(packet)
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
>>> ans
<Results: TCP:1 UDP:0 ICMP:0 Other:0>
>>> ans[0]
QueryAnswer(query=<IP  frag=0 proto=tcp dst=192.168.84.128 |<TCP  dport=http |<Raw  load='Hello World' |
>>>, answer=<IP  version=4 ihl=5 tos=0x0 len=44 id=0 flags=DF frag=0 ttl=64 proto=tcp chksum=0x1079 src=
192.168.84.128 dst=192.168.84.130 |<TCP  sport=http dport=ftp_data seq=1280939455 ack=1 dataofs=6 reserv
ed=0 flags=SA window=5840 chksum=0x7475 urgptr=0 options=[('MSS', 1460)] |<Padding  load='\x00\x00' |>>>
)
>>> ans[0][0]
<IP  frag=0 proto=tcp dst=192.168.84.128 |<TCP  dport=http |<Raw  load='Hello World' |>>>
>>> 
```

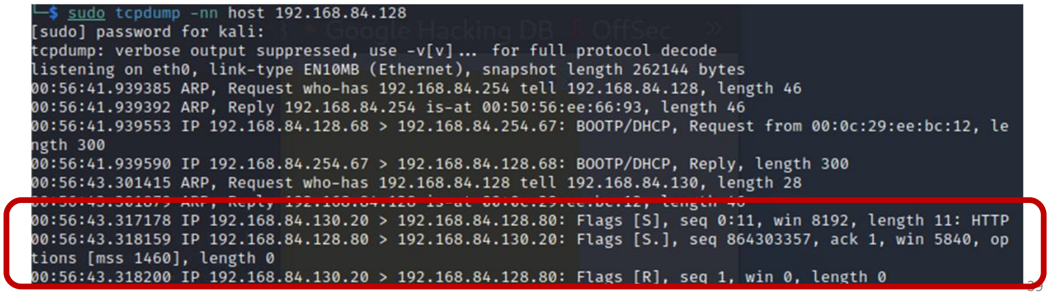
# Reading and Writing Packets

- To get packet from a pcap (packet capture) file
  - ❖>>> rdpcap("filename")
- To write packets to a file
  - ❖>>> wrpcap("filename", packets)
- To view packets in Wireshark
  - ❖>>> **wireshark(packets)**

# Inspecting Results



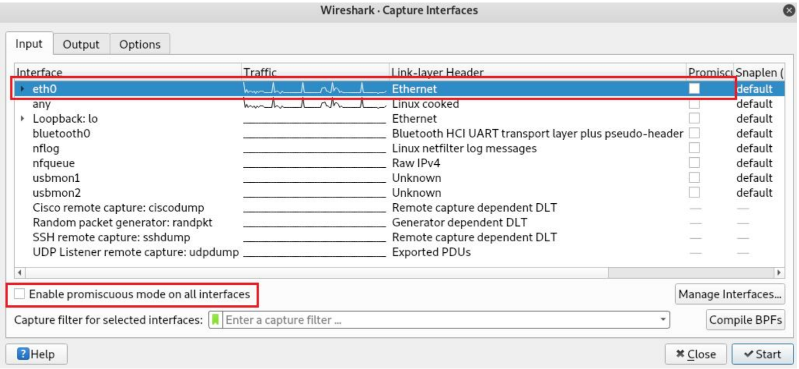
```
>>> packet=IP(dst="192.168.84.128")/TCP(dport=80)/"Hello World"
>>> sr(packet)
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
(<Results: TCP:1 UDP:0 ICMP:0 Other:0>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>> 
```



```
└─$ sudo tcpdump -nn host 192.168.84.128
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
00:56:41.939385 ARP, Request who-has 192.168.84.254 tell 192.168.84.128, length 46
00:56:41.939392 ARP, Reply 192.168.84.254 is-at 00:50:56:ee:66:93, length 46
00:56:41.939553 IP 192.168.84.128.68 > 192.168.84.254.67: BOOTP/DHCP, Request from 00:0c:29:ee:bc:12, le
ngth 300
00:56:41.939590 IP 192.168.84.254.67 > 192.168.84.128.68: BOOTP/DHCP, Reply, length 300
00:56:43.301415 ARP, Request who-has 192.168.84.128 tell 192.168.84.130, length 28
00:56:43.301879 ARP, Reply 192.168.84.128 is-at 00:0c:29:ee:bc:12, length 46
00:56:43.317178 IP 192.168.84.130.20 > 192.168.84.128.80: Flags [S], seq 0:11, win 8192, length 11: HTTP
00:56:43.318159 IP 192.168.84.128.80 > 192.168.84.130.20: Flags [S.], seq 864303357, ack 1, win 5840, op
tions [mss 1460], length 0
00:56:43.318200 IP 192.168.84.130.20 > 192.168.84.128.80: Flags [R], seq 1, win 0, length 0
```

# Capturing Traffic Using Wireshark

- Promiscuous mode
  - ❖In Wireshark, Capture >> Options



| Interface | Traffic | Link-layer Header | Promiscu | Snaplen ( |
|---|---|---|---|---|
| ▸ eth0 | | Ethernet | □ | default |
| any | | Linux cooked | □ | default |
| ▸ Loopback: lo | | Ethernet | □ | default |
| bluetooth0 | | Bluetooth HCI UART transport layer plus pseudo-header | □ | default |
| nflog | | Linux netfilter log messages | □ | default |
| nfqueue | | Raw IPv4 | □ | default |
| usbmon1 | | Unknown | □ | default |
| usbmon2 | | Unknown | □ | default |
| Cisco remote capture: ciscodump | | Remote capture dependent DLT | — | — |
| Random packet generator: randpkt | | Generator dependent DLT | — | — |
| SSH remote capture: sshdump | | Remote capture dependent DLT | — | — |
| UDP Listener remote capture: udpdump | | Exported PDUs | — | — |

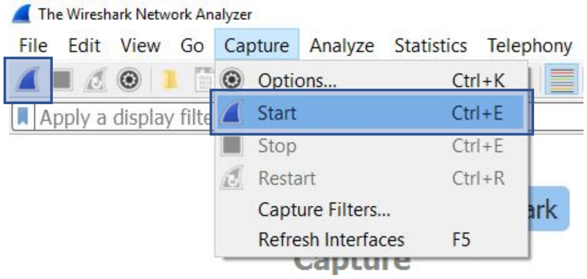
□ Enable promiscuous mode on all interfaces

Capture filter for selected interfaces: Enter a capture filter ...

46

# Using Wireshark

- Capturing Traffic



47

47

# Wireshark – Filtering Traffic

- Comparison operators
  - ❖eq , ==
  - ❖ne , !=
  - ❖gt , >
  - ❖lt , <
  - ❖ge , >=
  - ❖le , <=

e.g) ip.addr==192.168.84.128

- Search and match operators
  - ❖contains Does the protocol, field, or slice contain a value
  - ❖matches Does the protocol or text string match the given Perl regular expression
  - ❖Follow >> TCP stream >> Search!

48

48

# Wireshark - Filtering Traffic

- Functions

| | |
|---|---|
| ❖upper(string-field) | convert a string field to uppercase |
| ❖lower(string-field) | convert a string field to lowercase |

- Protocol field types
  - ❖http.host
  - ❖tcp.port
  - ❖ip.src
  - ❖ip.dst
  - ❖eth.addr

- **protocol.field operator value**

# Wireshark - Filtering Traffic

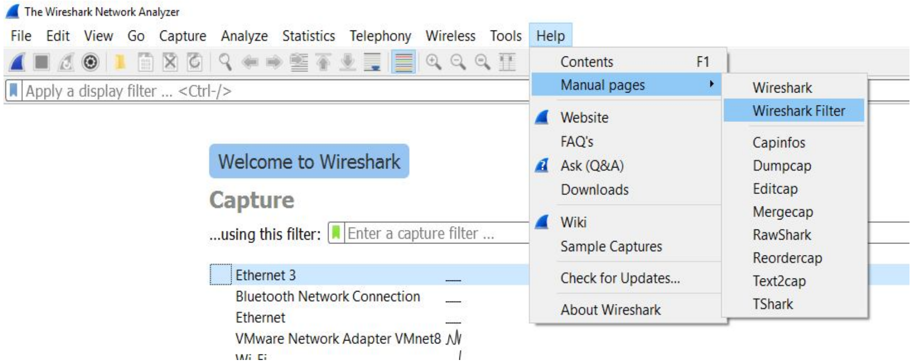
- The slice operator

| | |
|---|---|
| ❖[i:j] | i = start_offset, j = length |
| ❖[i-j] | i = start_offset, j = end_offset, inclusive |
| ❖[i] | i = start_offset, length = 1 |
| ❖[:j] | start_offset = 0, length = j |
| ❖[i:] | start_offset = i, end_offset = end_of_field |

- Membership operator
  - ❖tcp.port in { <port numbers> }

# Wireshark – Filtering Traffic

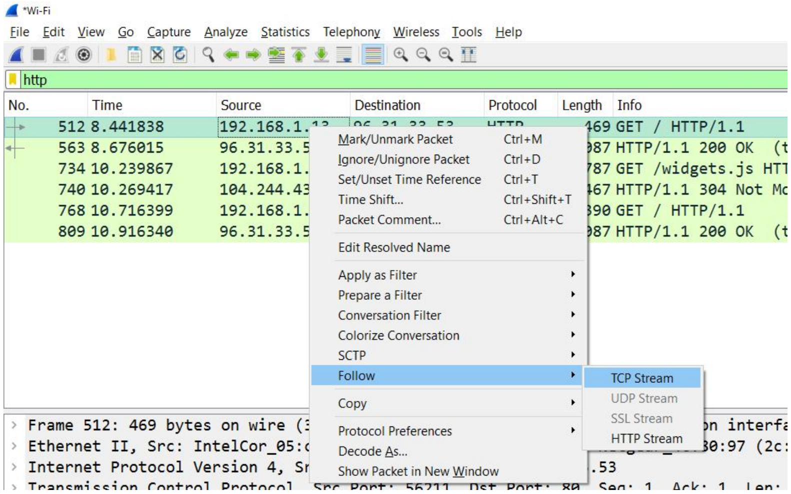
- More filtering information



51

51

# Wireshark – Following a TCP Stream


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Mark/Unmark Packet Ctrl+M
Ignore/Unignore Packet Ctrl+D
Set/Unset Time Reference Ctrl+T
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Edit Resolved Name
Apply as Filter
Prepare a Filter
Conversation Filter
Colorize Conversation
SCTP
Follow
Copy
Protocol Preferences
Decode As...
Show Packet in New Window
TCP Stream
UDP Stream
SSL Stream
HTTP Stream

http

```
No.     Time       Source        Destination   Protocol  Length Info
    512 8.441838   192.168.1.13  96.31.33.53   HTTP      469 GET / HTTP/1.1
    563 8.676015   96.31.33.5                            987 HTTP/1.1 200 OK  (t
    734 10.239867  192.168.1.                            787 GET /widgets.js HTT
    740 10.269417  104.244.43                            467 HTTP/1.1 304 Not Mo
    768 10.716399  192.168.1.                            390 GET / HTTP/1.1
    809 10.916340  96.31.33.5                            987 HTTP/1.1 200 OK  (t

> Frame 512: 469 bytes on wire (3                              on interfa
> Ethernet II, Src: IntelCor_05:c                              30:97 (2c:
> Internet Protocol Version 4, Sr                              .53
> Transmission Control Protocol, Src Port: 56211, Dst Port: 80, Seq: 1, Ack: 1, Len:
```

52

52

# ARP Cache Poisoning

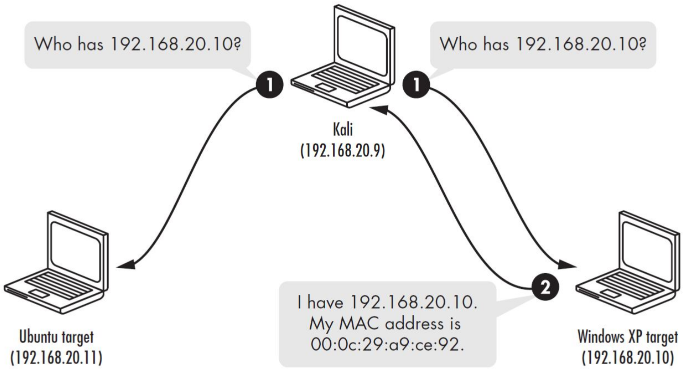
- Let's see the traffic that wasn't intended for our Kali system **for pentesting purposes**
- Network switch will send only packets that belong to us
  - ❖We need to trick our target machine or the switch (or ideally both) into believing the traffic belongs to us
- Man-in-the-middle (MITM) attack
  - ❖Allow us to redirect and intercept traffic between two systems
  - ❖Address Resolution Protocol (ARP) cache poisoning
  - ❖Also known as ARP spoofing
    - ❖Attack on **Confidentiality**

# ARP Basics

- Switch forwards a packet based on its MAC address
- Sender broadcasts "who has the IP address 192.168.19.129?"
- The machine with 192.168.19.129 responds, "I'm 192.168.19.129 and my MAC address is 11:22:33:44:55:66."
- # arp –a
  - Displays the current arp cache

# ARP Resolution Process

**Man-In-The-Middle (MITM)**



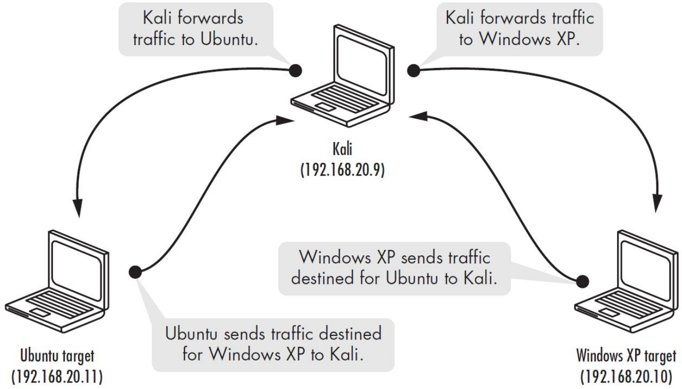
55

55

# ARP Cache Poisoning

**1) IP forwarding**
**2) Arp cache poisoning**

**Man-In-The-Middle (MITM)**



56

56

# IP Forwarding

- Forwards any extraneous packets it receives to their proper destination

- $ sudo echo **1** > /proc/sys/net/ipv4/ip_forward
  - After this setting, Kali will forward irrelevant packet to the right destination

- $ sudo sysctl net.ipv4.ip_forward=1
  - Same function..

# ARP Cache Poisoning with Arpspoof

- There is no guarantee (checking mechanism) that the IP address to MAC address answer you get is correct

- To fool the target machine into thinking we are the authentic receiver:
  - ❖# arpspoof -i eth0 -t <target1 IP> <target2 IP>
  - ❖-i: specify the interface
  - ❖-t: specify the target IP addresses

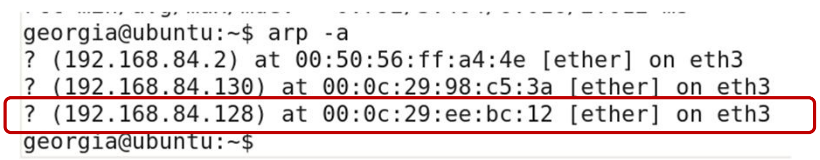
"Hello, Target1! I'm Target2"

  - # arpspoof -i etho -t <target2 IP> <target1 IP>
    - ❖Connection should be bi-directional so it can be stealthy

"Hello, Target2! I'm Target1"

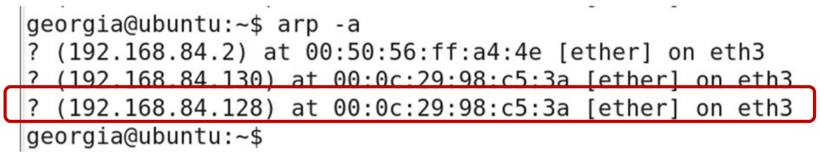
# ARP Poisoning

- Before ARP poisoning



```
georgia@ubuntu:~$ arp -a
? (192.168.84.2) at 00:50:56:ff:a4:4e [ether] on eth3
? (192.168.84.130) at 00:0c:29:98:c5:3a [ether] on eth3
? (192.168.84.128) at 00:0c:29:ee:bc:12 [ether] on eth3
georgia@ubuntu:~$
```

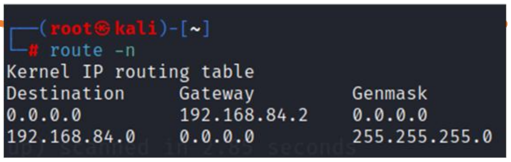
- After ARP poisoning



```
georgia@ubuntu:~$ arp -a
? (192.168.84.2) at 00:50:56:ff:a4:4e [ether] on eth3
? (192.168.84.130) at 00:0c:29:98:c5:3a [ether] on eth3
? (192.168.84.128) at 00:0c:29:98:c5:3a [ether] on eth3
georgia@ubuntu:~$
```

# Impersonate the Default Gateway

- To find the default gateway, type
  - ❖# route -n



```
┌──(root㉿kali)-[~]
└─# route -n
Kernel IP routing table
Destination     Gateway         Genmask
0.0.0.0         192.168.84.2    0.0.0.0
192.168.84.0    0.0.0.0         255.255.255.0
```

- We can also use ARP cache poisoning to impersonate the default gateway on a network and access traffic entering and leaving the network, including traffic destined for the Internet

- NOTE
  - ❖If you use ARP cache poisoning to trick a large network into thinking your pentest machine is the default gateway, you may unwittingly cause networking issues. All the traffic in a network going through one laptop can slow things down to the point of denial of service in some cases

# DNS Cache Poisoning

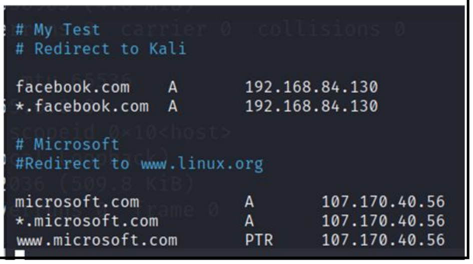
- DNS maps (or resolves) domain names to IP addresses
  - ❖DNS resolution translates the human-readable domain name into an IP address
- # nslookup www.youtube.com
- Like ARP cache poisoning, we can poison Domain Name Service (DNS) cache entries to route traffic intended for another website to one we control
  - ❖We send a bunch of bogus **DNS resolution replies** pointing to the wrong IP address for a domain name

# Ettercap

- Ettercap is a free and open source network security tool for **man-in-the-middle attacks** on LAN
- Runs on Linux and Windows
- It is capable of intercepting traffic on a network segment, capturing passwords, and conducting active eavesdropping against a number of common protocols
- Ettercap works by putting the network interface into promiscuous mode and by ARP poisoning the target machines

## DNS Cache Poisoning Using Ettercap

- # echo 1 > /proc/sys/net/ipv4/ip_forward (IP forwarding enabled)
- # locate etter.dns
- # gedit /etc/ettercap/etter.dns
  - ❖Put a host information
    - facebook.com A <Kali IP address>
    - *.facebook.com A <Kali IP address>

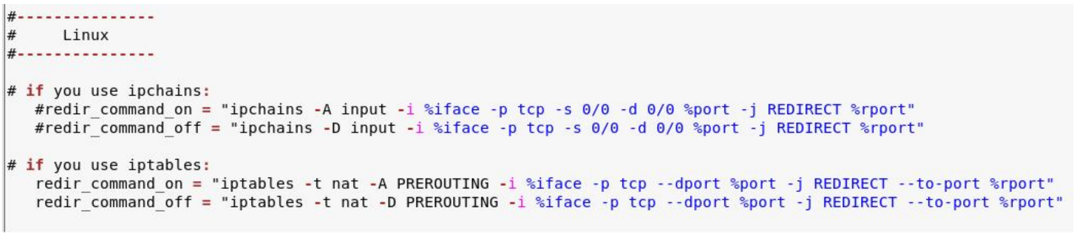


```
# My Test
# Redirect to Kali

facebook.com    A       192.168.84.130
*.facebook.com  A       192.168.84.130

# Microsoft
#Redirect to www.linux.org

microsoft.com           A       107.170.40.56
*.microsoft.com         A       107.170.40.56
www.microsoft.com       PTR     107.170.40.56
```

63

## DNS Cache Poisoning Using Ettercap

- $ sudo gedit /etc/ettercap/etter.conf
  - ❖Uncomment (removing #) redir_command_on [off] under "if you use iptables:"
- $sudo service apache2 start
  - ❖Start your web server on Kali

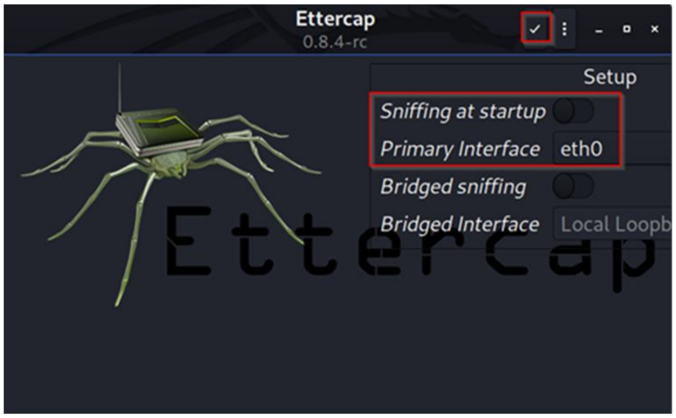


```
#---------------
#     Linux
#---------------

# if you use ipchains:
   #redir_command_on = "ipchains -A input -i %iface -p tcp -s 0/0 -d 0/0 %port -j REDIRECT %rport"
   #redir_command_off = "ipchains -D input -i %iface -p tcp -s 0/0 -d 0/0 %port -j REDIRECT %rport"

# if you use iptables:
   redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp --dport %port -j REDIRECT --to-port %rport"
   redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp --dport %port -j REDIRECT --to-port %rport"
```

64

# DNS Cache Poisoning Using Ettercap

- $ sudo ettercap -G



Sniffing at startup ○

Primary Interface eth0

Bridged sniffing ○

Bridged Interface Local Loopb