

2. Network Traffic & Flow Analysis

Data Link Layer

Ethernet 802.3

ARP Attacks & Detection

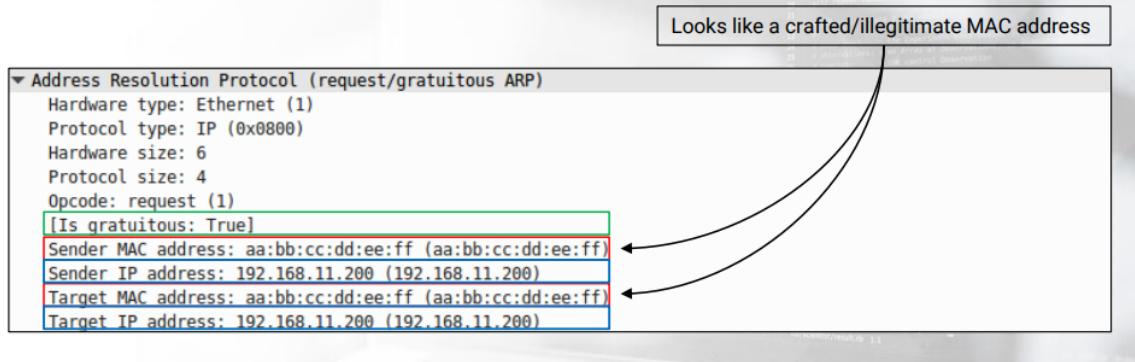
- The so-called gratuitous ARP requests and responses are also possible, and they are usually abused by attackers.
- Gratuitous ARP request:** It is a request packet where **the source and destination IP are set with the IP of the machine** that is issuing the packet and the **destination MAC is the broadcast address**.
- Gratuitous ARP may be useful to detect IP conflict or simply inform other hosts/switches of a MAC address in the network, but attackers can also use these packets to mount **ARP poisoning attacks**.

Normal Gratuitous ARP

Make sure the MAC address is legit / belongs to your organization

Address Resolution Protocol (request/gratuitous ARP)	
Hardware type: Ethernet (1)	
Protocol type: IP (0x0800)	
Hardware size: 6	
Protocol size: 4	
Opcode: request (1)	
[Is gratuitous: True]	
Sender MAC address: 00:50:56:c0:00:01 (00:50:56:c0:00:01)	
Sender IP address: 192.168.11.200 (192.168.11.200)	
Target MAC address: 00:50:56:c0:00:01 (00:50:56:c0:00:01)	
Target IP address: 192.168.11.200 (192.168.11.200)	

Attacker-crafted Gratuitous ARP



- **Gratuitous ARP reply:** It is an ARP reply that has been sent **without being requested**. (usually malicious)

Tips regarding normal and suspicious ARP traffic.

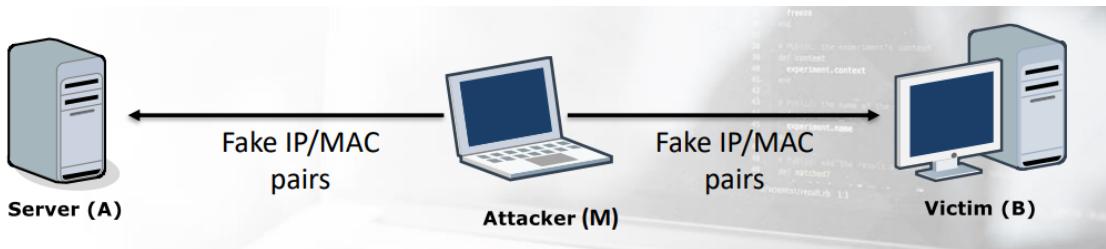
- **Normal:** ARP broadcasts are normal from both clients and servers, including network devices at a reasonable flow.
- **Suspicious:** Tens, hundreds, or even thousands of ARP broadcast messages within a small time window

No.	Time	Source	Destination	Protocol	Length	Info
15	61.162590056	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.1? Tell 172.16.5.67
16	61.164533730	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.2? Tell 172.16.5.67
17	61.166589500	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.3? Tell 172.16.5.67
18	61.171696684	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.4? Tell 172.16.5.67
19	61.173595193	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.5? Tell 172.16.5.67
20	61.175482595	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.6? Tell 172.16.5.67
21	61.177434405	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.7? Tell 172.16.5.67
22	61.179428423	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.8? Tell 172.16.5.67
23	61.181401311	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.9? Tell 172.16.5.67
24	61.183387692	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.10? Tell 172.16.5.67
25	61.185470650	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.11? Tell 172.16.5.67
26	61.187379238	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.12? Tell 172.16.5.67
27	61.189625522	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.13? Tell 172.16.5.67
28	61.191455492	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.14? Tell 172.16.5.67
29	61.193387656	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.15? Tell 172.16.5.67
30	61.195423342	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.16? Tell 172.16.5.67
31	61.197387752	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.17? Tell 172.16.5.67
32	61.199389322	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.18? Tell 172.16.5.67
33	61.201395568	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.19? Tell 172.16.5.67
34	61.203388474	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.20? Tell 172.16.5.67

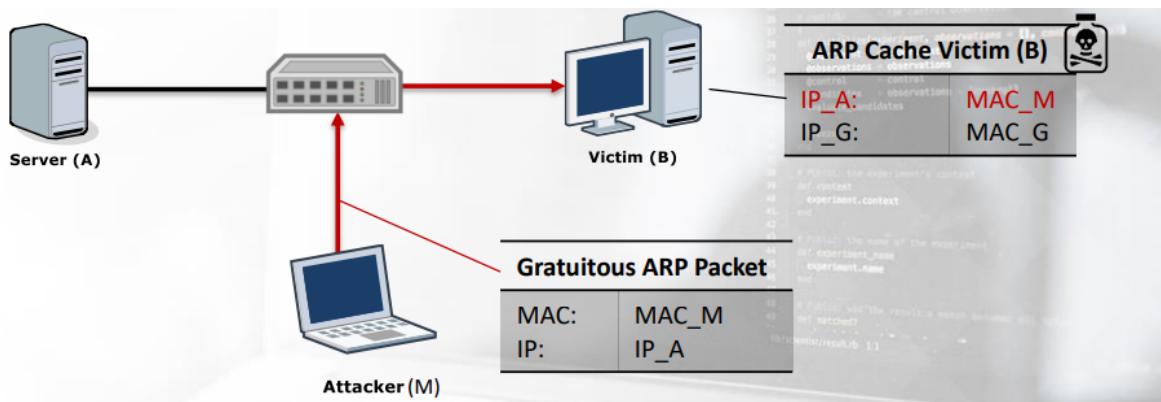
ARP poisoning (between two communication peers into a local network)

- ARP poisoning can be exploited to add fake information between
 - The following are the steps for a successful attack (**ARP Poisoning**):
1. **M** would pretend to be **B to A**: it will send a **gratuitous ARP reply** with the pair: **IP_B->MAC_M**

2. M would pretend to be **A to B**: it will send a **gratuitous ARP reply** with the pair:
IP_A->MAC_M



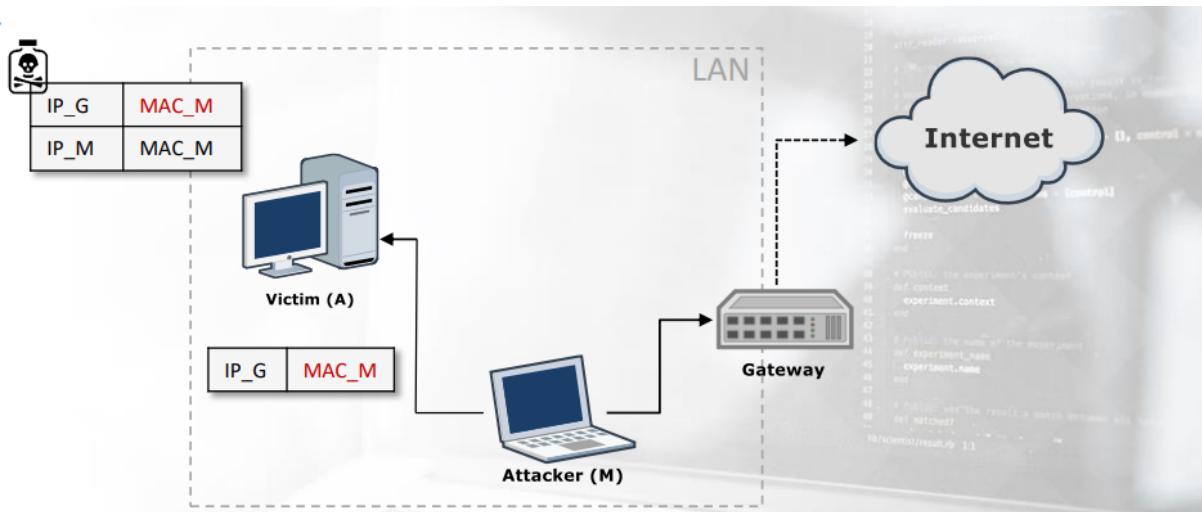
- Because of the **TTL** in hosts **ARP caches**, an attacker would need to send these packets at intervals lower than the timeout (usually every **30** seconds is a good choice).
- Once the gratuitous ARP packet is sent, B's ARP cache gets poisoned with the entry: **IP_A->MAC_M**. Next time B wants to send a packet to A, it will be forwarded to M. (The same thing happens against A.)



- Another gratuitous ARP with correct values would restore the correct values after the sniffing is completed

ARP poisoning (between local host and remote host)

- When a host in a LAN wants to send packets to hosts outside the LAN, it uses the **default gateway**.
- The default gateway MAC address must be used to **forward** the packet along with the correct IP address configured by the administrator or given by DHCP.
- The use of ARP poisoning in this scenario leads to a **MITM attack from local to remote**.



The following describes the steps that take place in the previous scenario:

1. **Host A wants to send packets to the Internet.** It already has the IP of the gateway (IP_G), and it needs the associated MAC address.
2. M can use a **gratuitous ARP reply** to advertise itself as the default gateway: **binds IP_G with his own (MAC_M).**
3. All the traffic meant to leave the LAN will pass **through M(the attacker)**, which will then redirect it to the real gateway.

ARP Spoofing Prevention

1. Using Static ARP
 - not a feasible approach into large and always-changing networks
2. Tools like **arpwatch** can detect but not stop such attacks
3. **Switches** usually feature protections against ARP spoofing (Port Security)

2.4.6.1 Checking the ARP Cache

You can check the ARP cache of your host by typing:

- arp -a on Windows.
- arp on *nix operating systems
- ip neighbour on Linux

```
$ ip neighbour
192.168.17.202 dev eth0 lladdr d0:d4:12:e1:ef:5a STALE
192.168.17.1 dev eth0 lladdr 00:50:7f:78:fc:40 STALE
192.168.17.99 dev eth0 lladdr 00:d0:4b:92:2d:89 STALE
192.168.17.14 dev eth0 lladdr 60:a4:4c:a8:be:5b STALE
192.168.17.18 dev eth0 lladdr 20:cf:30:c7:ad:ae STALE
192.168.17.30 dev eth0 lladdr 20:cf:30:ea:22:13 STALE
192.168.17.66 dev eth0 lladdr a4:ee:57:e8:2e:0b STALE
192.168.17.254 dev eth0 lladdr c8:4c:75:a4:79:a6 REACHABLE
192.168.17.12 dev eth0 lladdr 60:a4:4c:a8:bd:1a STALE
192.168.17.19 dev eth0 lladdr 54:04:a6:a0:6e:ad STALE
192.168.17.24 dev eth0 lladdr bc:5f:f4:ef:63:51 STALE
```

MAC Flooding

- switches store the MAC address to physical switch port pairing in their **Content Addressable Memory (CAM)** table.
 - <MAC address - port number - TTL>.
- **MAC flooding is meant to fill the CAM table of the switch.**
- When the space in the CAM is **filled** with fake MAC addresses, the **switch cannot learn new MAC addresses** so it forces switches to behave like a **hub** and then **forward frames on all the ports**.

MAC Flooding Prevention

- **port security** (restrict the association of a port with a single source MAC address)
- Additionally, there are switches that can be configured in such a way so that acting like a hub is prohibited.

802.11 Wireless (layer 2) header

The types of 802.11 packets are:

- **Management:** Connectivity between hosts at layer 2 is based upon those packets.
 - Authentication packets

- Association packets
- **Beacon** packets
- **Control:** Delivery of packets is enabled by those packets. Congestion is also “regulated” by them.
 - Request-to-send packets
 - Clear-to-send packets
- **Data:** Those packets are the actual data containers. They are the only packet kind that can be passed from the wireless to the wired network.
- **Beacon packets** are broadcasted from a wireless access point to inform other listening wireless clients of its existence and its connection requirements.

The 802.11 management frame header contains information such as:

- **Timestamp:** Packet transmission time
- **Beacon Interval:** Beacon packet retransmission time
- **Capabilities Information:** Hardware capabilities of the AP
- **SSID parameter set:** Network name broadcasted by the AP
- **Supported Rates:** Data transfer rates supported by the AP
- **DS Parameter set:** Channel on which the AP operates

```

Frame 1: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits)
  ▾ IEEE 802.11 Beacon frame, Flags: .....
    ▶ Type/Subtype: Beacon frame (0x0008)
    ▶ Frame Control Field: 0x8000
      .000 0000 0000 = Duration: 0 microseconds
      Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
      Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
      Transmitter address: D-Link_0b:22:ba (00:13:46:0b:22:ba)
      Source address: D-Link_0b:22:ba (00:13:46:0b:22:ba)
      BSS Id: D-Link_0b:22:ba (00:13:46:0b:22:ba)
      .... .... 0000 = Fragment number: 0
      0101 0100 1000 .... = Sequence number: 1352
  ▾ IEEE 802.11 wireless LAN management frame
    ▾ Fixed parameters (12 bytes)
      ▶ Timestamp: 0x000000001685a181
      ▶ Beacon Interval: 0.102400 [Seconds]
      ▶ Capabilities Information: 0x0431
    ▾ Tagged parameters (96 bytes)
      ▶ Tag: SSID parameter set: [REDACTED]
      ▶ Tag: Supported Rates 1(B), 2(B), 5.5(B), 11(B), 6, 12, 24, 36, [Mbit/sec]
      ▶ Tag: DS Parameter set: Current Channel: 11
      ▶ Tag: Traffic Indication Map (TIM): DTIM 0 of 0 bitmap
      ▶ Tag: ERP Information
      ▶ Tag: Extended Supported Rates 9, 18, 48, 54, [Mbit/sec]
      ▶ Tag: Vendor Specific: AtherosC: Advanced Capability
      ▶ Tag: Vendor Specific: AtherosC: Unknown
      ▶ Tag: Vendor Specific: AtherosC: eXtended Range
      ▶ Tag: Vendor Specific: GlobalSu
  
```

IP Layer Attacks with each header

IDS/Firewall Evasion (**invalid IP version**)

- Oftentimes, attackers check the reactions of firewalls and IDS by crafting and sending datagrams with an **invalid IP version**

No.	Time	Source	src port	dst pc	Destr	Protocol	Length	Info
1	0.000000	VMware_d5:f7...		V...	IPv4	1257	Bogus IPv4 version (7, must be 4)	
2	0.003648	VMware_d5:f7...		V...	IPv4	1059	Bogus IPv4 version (14, must be 4)	
3	0.007229	VMware_d5:f7...		V...	IPv4	1069	Bogus IPv4 version (10, must be 4)	
4	0.032550	VMware_d5:f7...		V...	IPv4	254	Bogus IPv4 version (10, must be 4)	
5	0.315966	VMware_d5:f7...		V...	IPv4	1271	Bogus IPv4 version (13, must be 4)	
6	0.320837	VMware_d5:f7...		V...	IPv4	1269	Bogus IPv4 version (9, must be 4)	
7	0.441523	VMware_d5:f7...		V...	IPv4	136	Bogus IPv4 version (0, must be 4)	
8	0.449408	VMware_d5:f7...		V...	IPv4	120	Bogus IPv4 version (2, must be 4)	
9	0.511513	VMware_d5:f7...		V...	IPv4	1237	Bogus IPv4 version (5, must be 4)	
10	0.520343	VMware_d5:f7...		V...	IPv4	177	Bogus IPv4 version (13, must be 4)	
11	0.606816	VMware_d5:f7...		V...	IPv4	1147	Bogus IPv4 version (0, must be 4)	

> Frame 6: 1269 bytes on wire (10152 bits), 1269 bytes captured (10152 bits)
> Ethernet II, Src: VMware_d5:f7:aa (00:0c:29:d5:f7:aa), Dst: VMware_a3:eb:77 (00:0c:29:a3:eb:77)
Internet Protocol Version 4
 < 1001 = Version: 9
 > [Expert Info (Error/Protocol): Bogus IPv4 version]

Version (p.version), 1 byte

Packets: 999 · Displayed: 999 (100.0%) Profile: Default

Stealthy Nmap Scan (changing the IPv4 Protocol Number.)

<https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>

- The venerable Nmap network scanner leverages this to perform IP protocol scanning against a given target.
- This type of scanning is also a stealthier way to identify a live host.

No.	Time	Source	src pc	dst pc	Destination	Protocol	Length	Info
16	0.000712	192.168.1.6		192.168.1.4		IPv4	60	
17	0.005358	192.168.1.6		192.168.1.4		IPv4	60	
18	0.005368	192.168.1.6		192.168.1.4		IPv4	60	
19	0.005371	192.168.1.6		192.168.1.4		IPv4	60	
20	0.005373	192.168.1.6		192.168.1.4		IPv4	60	
21	0.005423	192.168.1.6		192.168.1.4		IPv4	60	
22	0.005442	192.168.1.6		192.168.1.4		IPv4	60	
23	0.005500	192.168.1.6		192.168.1.4		IPv4	60	

> Frame 21: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
> Ethernet II, Src: VMware_d5:f7:aa (00:0c:29:d5:f7:aa), Dst: VMware_a3:eb:77 (00:0c:29:a3:eb:77)
Internet Protocol Version 4, Src: 192.168.1.6, Dst: 192.168.1.4
 < 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 20
 Identification: 0x71bd (29117)
 > Flags: 0x00
 Fragment Offset: 0
 Time to Live: 54
 Protocol: Unassigned (200)

Protocol (p.proto), 1 byte

Packets: 307 · Displayed: 307 (100.0%) Profile: Default

The screenshot shows a Wireshark interface with the following details:

- Packets:** 307
- Displayed:** 307 (100.0%)
- Profile:** Default

In the details pane, the highlighted section shows:

```

> Frame 22: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
> Ethernet II, Src: VMware_d5:f7:aa (00:0c:29:d5:f7:aa), Dst: VMware_a3:eb:77 (00:0c:29:a3:eb:77)
< Internet Protocol Version 4, Src: 192.168.1.6, Dst: 192.168.1.4
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 20
    Identification: 0x680f (26639)
    Flags: 0x00
    Fragment Offset: 0
    Time to Live: 49
    Protocol: Unknown (245)
  
```

Source & Destination IP Addresses

- There are three golden detection rules that are related to this set of IPv4 header fields.
1. **Incoming traffic** to your network should obviously have a **Source IP Address that doesn't belong to your network address space**. If it does, it is most probably **crafted**.
 2. **Outgoing traffic** from your network should obviously have a **Source IP Address that belongs to your network address space**. If it doesn't, there is most probably a misconfiguration or the address is **spoofed**.
 3. Private network addresses or the loopback mode address also require your attention

Fragmentation (abused for IDS/IPS evasion purposes)

- Fragmentation is the action of **dividing a packet** whose size is greater than the Maximum Transmission Unit (MTU)
- it can be performed by a **router** or the sending **host** itself.
- Each fragment's IP header contains fields and values that facilitate **reassembling** the original packet at the destination.
- When it comes to fragmented packets, IDS/IPS must act just if they were the **destination host**, in terms of **packet reassembling**. This is for obvious reasons, IDS/IPS need the **whole packet in order to inspect it**

- attackers can introduce difficulties in the reassembling procedure by the IDS/IPS, such as:
 - Crafted fragmented packets with **identical offsets** but **different payloads**
 - Crafted packets arriving with a **great time difference**
- For the IDS/IPS to safely perform such packet reassembling and inspection, it should act just like the destination host does. (wait as long as the destination does for a fragment to arrive)
- IP packet exceeding the 65535 bytes limit of data via a ping command (**ping-of-death**)

IPv6

IPv6 Tunneling

- It is a known fact that attackers have been using tunnelbased IPv6 transition mechanisms for hide communication and stealthy exfiltration over an IPv4-only or dual-stack network.

```

Apply a display filter ... <Ctrl-/>
No. Time Source src port dst port Destination Protocol Length
6 9.829544 fe80::ffff:ffff:f... 55519 teredo ff02::2 ICMP... 103
7 9.956850 fe80:0:7274:696e:... teredo 55519 fe80::ffff:ff... ICMP... 159

Frame 7: 159 bytes on wire (1272 bits), 159 bytes captured (1272 bits)
Ethernet II, Src: VMware_e1:a9:f8 (00:50:56:e1:a9:f8), Dst: VMware_60:22:18 (00:0c:29:60:2...)
Internet Protocol Version 4, Src: 83.170.6.76, Dst: 192.168.73.148
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 145
        Identification: 0x7a20 (31264)
        Flags: 0x00
        Fragment Offset: 0
        Time to Live: 128
        Protocol: UDP (17)
        Header Checksum: 0x5c09 [validation disabled]
            [Header checksum status: Unverified]
        Source Address: 83.170.6.76
        Destination Address: 192.168.73.148
User Datagram Protocol, Src Port: teredo (3544), Dst Port: 55519 (55519)
Teredo IPv6 over UDP tunneling
    Teredo Authentication header
        Client identifier length: 0
        Authentication value length: 0
        Nonce value: 58b1b73fc3d0859b
        Confirmation byte: 00
    Teredo Origin Indication header
Internet Protocol Version 6, Src: fe80:0:7274:696e:8000:dd8:ac55:f9b3, Dst: fe80::ffff:fff...
    0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 56
    Next Header: ICMPv6 (58)
    Hop Limit: 255
    Source Address: fe80:0:7274:696e:8000:dd8:ac55:f9b3
    Teredo Authentication header (teredo.auth), 13 bytes
    Packets: 169 • Displayed: 169 (100.0%) | Profile: Default

```

Network Discovery Attacks

- Attackers ultimately want to introduce incorrect IPv6 host address/link layer pairings; this can be achieved via two (2) distinct ways:
 1. An attacker on the same local network can tamper with a returned **Neighbor Advertisement (NA)** spoofing an address, after a **Neighbor Solicitation (NS)** request is sent; this is the equivalent of **ARP poisoning in IPv4**.
 2. An attacker can also craft an **NS** request containing the fake IPv6 host address/link layer pairing. Listening neighbors will introduce this not requested pairing in their neighbor cache; this is the equivalent of **abused Gratuitous ARP in IPv4**.
- Other Network Discovery attacks include:
- Causing a **Dos**, by spoofing an **NA response**, informing the NS request sender that the target host resides at a non-existing link address. The same can be

achieved by abusing the Neighbor Unreachable Protocol to sent a spoofed NA response informing that communication with the target is not possible.

- Causing a **DoS**, by spoofing an **NS response**, informing that the address is taken. Recall the Duplicate Address Detection procedure. DAD could be abused multiple times to prevent a host from being assigned an address.
- Executing a **man-in-the-middle attack**, by spoofing an RA, informing the host that sent the RS message that the attacker's host is the router.
- Those, are only a subset of the attacks that can be executed against an IPv6 implementation. For more, please refer to the following resources:
 1. <https://www.ripe.net/support/training/material/ipv6-security/ipv6security-slides.pdf>
 2. <https://www.blackhat.com/docs/sp-14/materials/arsenal/sp-14-Schaefer-Workshop-Slides.pdf>
 3. https://www.tno.nl/media/3274/testing_the_security_of_ipv6_implementations.pdf

Transport Layer Attacks

▼ TCP - UDP Theoretical

Transport layer (segments/datagrams)

the transport layer is the link between the application layer and the lower layers that are responsible for network transmission.

Transport Layer Responsibilities:

1. Tracking Individual Conversations
2. Segmenting Data and Reassembling Segments
3. Add Header Information
4. Identifying the Applications
5. Conversation Multiplexing
 - a. the transport layer uses segmentation and multiplexing to enable different communication conversations to be interleaved on the same network.

the benefits of dividing the data into segments :

- **Increases speed** - Because a large data stream is segmented into packets, large amounts of data can be sent over the network without tying up a communications link. This allows many different conversations to be interleaved on the network called multiplexing.
- **Increases efficiency** - If a single segment fails to reach its destination due to a failure in the network or network congestion, only that segment needs to be retransmitted instead of resending the entire data stream.
- TCP is responsible for sequencing the individual segments.

TCP & UDP

TCP

- Transmission Control Protocol. Enables reliable communication between processes running on separate hosts and provides reliable, acknowledged transmissions that confirm successful delivery. (**segments**) (Connection-Oriented=must first establish a connection between the sender and the receiver)

TCP provides reliability and flow control using these basic operations:

- Number and track data segments transmitted to a specific host from a specific application
- Acknowledge received data
- Retransmit any unacknowledged data after a certain amount of time
- Sequence data that might arrive in wrong order
- Send data at an efficient rate that is acceptable by the receiver

TCP Header Fields



The table identifies and describes the ten fields in a TCP header.

TCP Header Field	Description
Source Port	A 16-bit field used to identify the source application by port number.
Destination Port	A 16-bit field used to identify the destination application by port number.
Sequence Number	A 32-bit field used for data reassembly purposes.
Acknowledgment Number	A 32-bit field used to indicate that data has been received and the next byte expected from the source.
Header Length	A 4-bit field known as "data offset" that indicates the length of the TCP segment header.
Reserved	A 6-bit field that is reserved for future use.
Control bits	A 6-bit field that includes bit codes, or flags, which indicate the purpose and function of the TCP segment.
Window size	A 16-bit field used to indicate the number of bytes that can be accepted at one time.
Checksum	A 16-bit field used for error checking of the segment header and data.
Urgent	A 16-bit field used to indicate if the contained data is urgent.

The six **control bits** flags are as follows:

- **URG** - Urgent pointer field significant
- **ACK** - Acknowledgment flag used in connection establishment and session termination
- **PSH** - Push function
- **RST** - Reset the connection when an error or timeout occurs
- **SYN** - Synchronize sequence numbers used in connection establishment
- **FIN** - No more data from sender and used in session termination

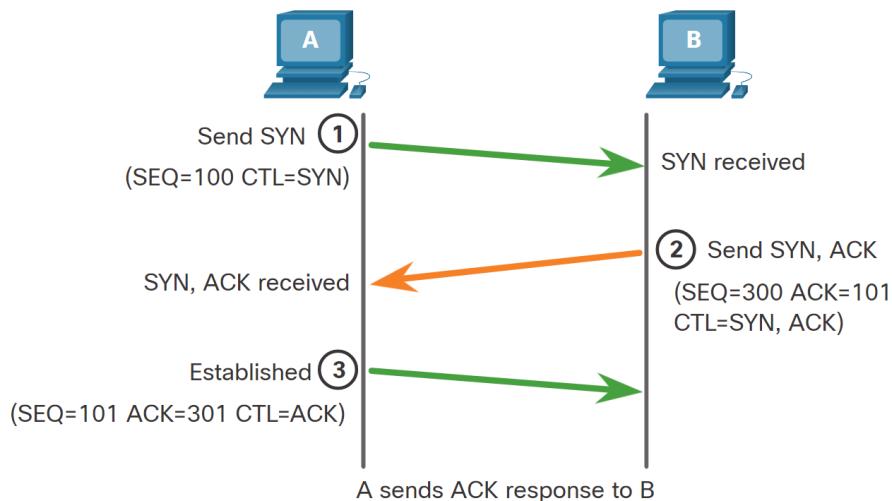
TCP Server Processes

- Each application process running on a server is configured to use a port number.
- The port number is either automatically assigned or configured manually by a system administrator.
- An individual server cannot have two services assigned to the same port number within the same transport layer services.

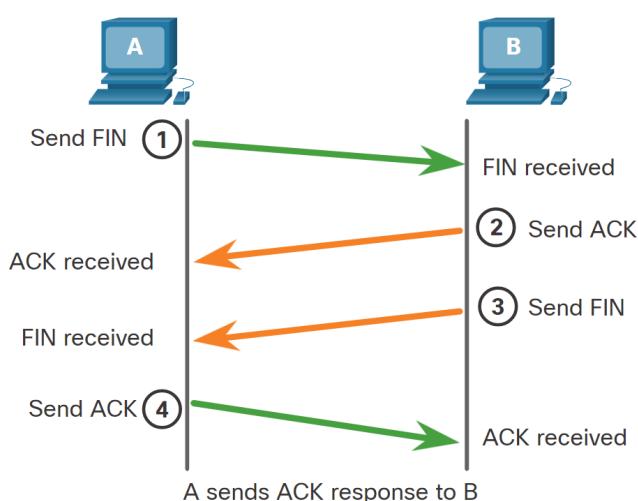
- There can be many ports open simultaneously on a server, one for each active server application.

TCP Connection Establishment (3-way Handshake)

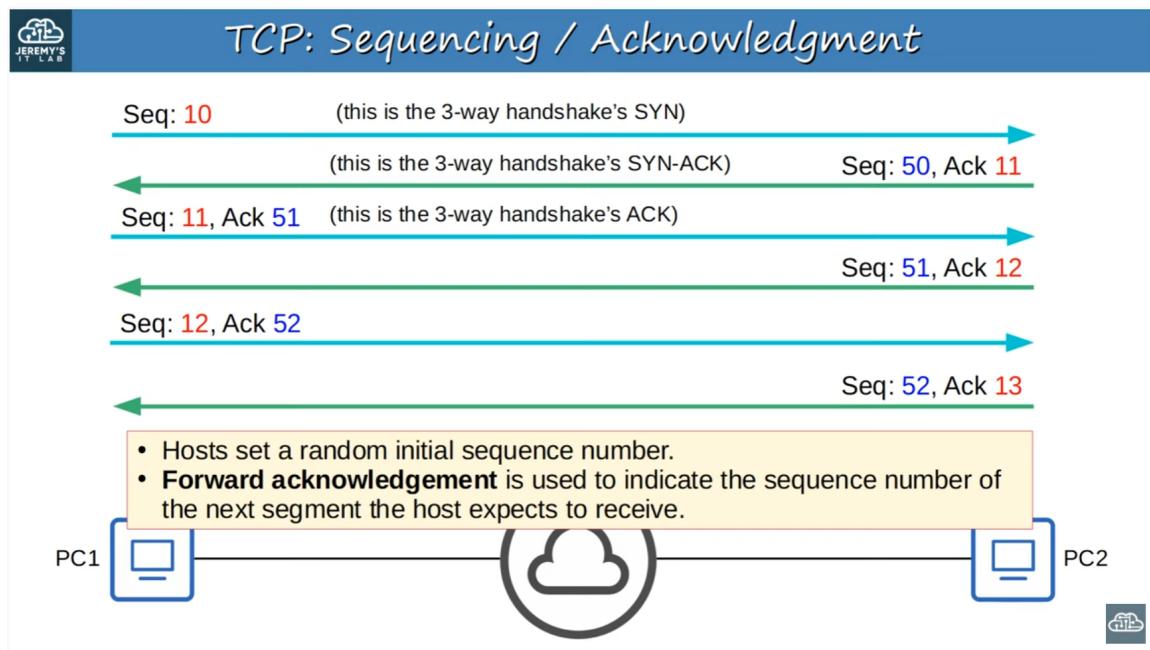
- It establishes that the destination device is present on the network.
- It verifies that the destination device has an active service and is accepting requests on the destination port number that the initiating client intends to use.
- It informs the destination device that the source client intends to establish a communication session on that port number.



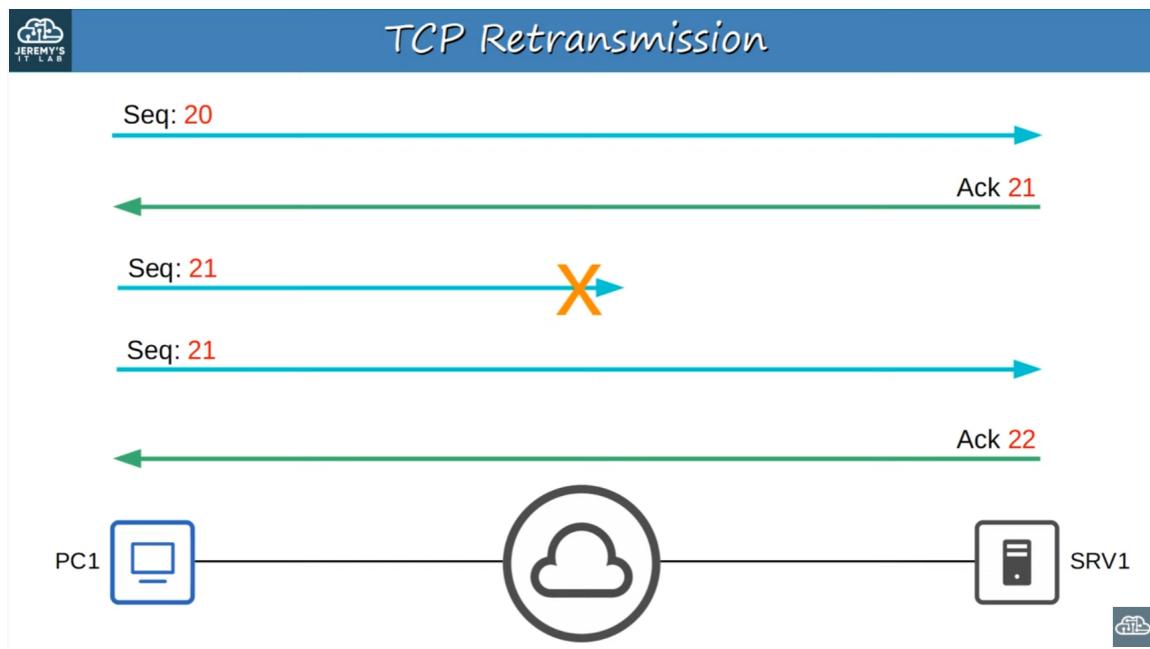
Session Termination (4-way Handshake)



TCP Reliability – Sequence Numbers and Acknowledgements



TCP Retransmission

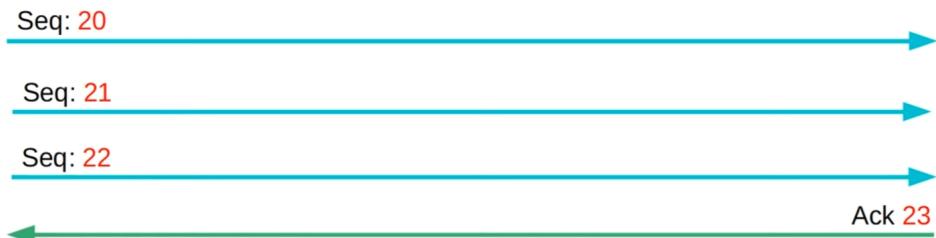


TCP Flow Control - Window Size (Watch)



TCP Flow Control: Window Size

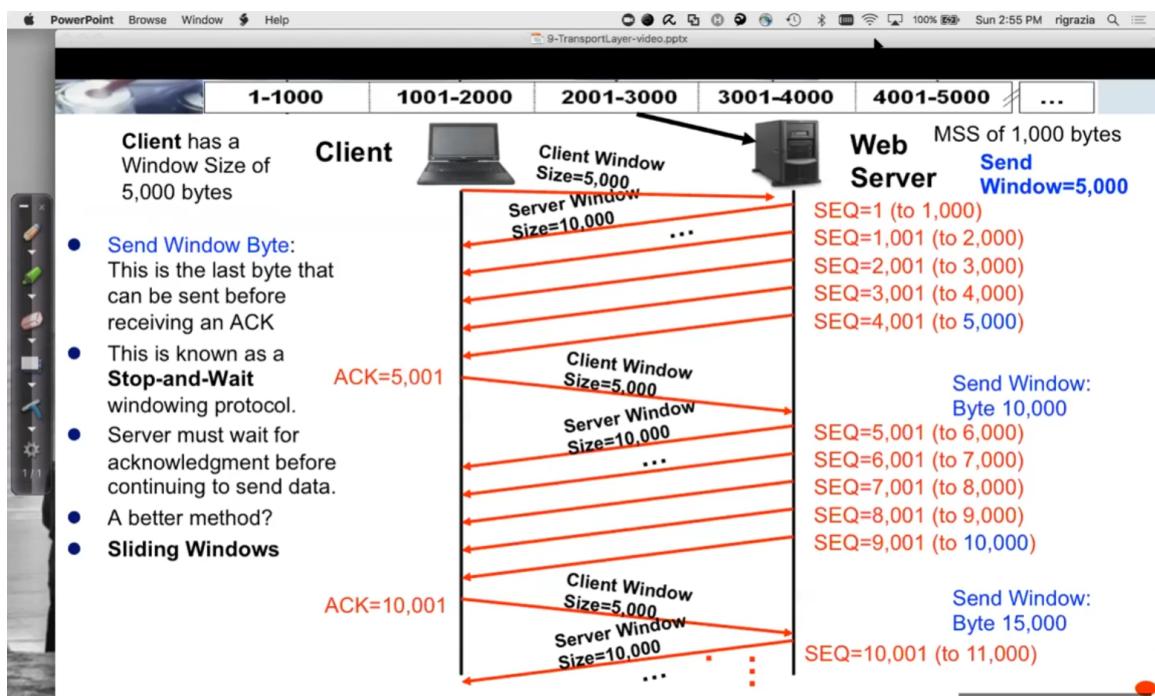
- Acknowledging every single segment, no matter what size, is inefficient.
- The TCP header's **Window Size** field allows more data to be sent before an acknowledgment is required.
- A 'sliding window' can be used to dynamically adjust how large the window size is.



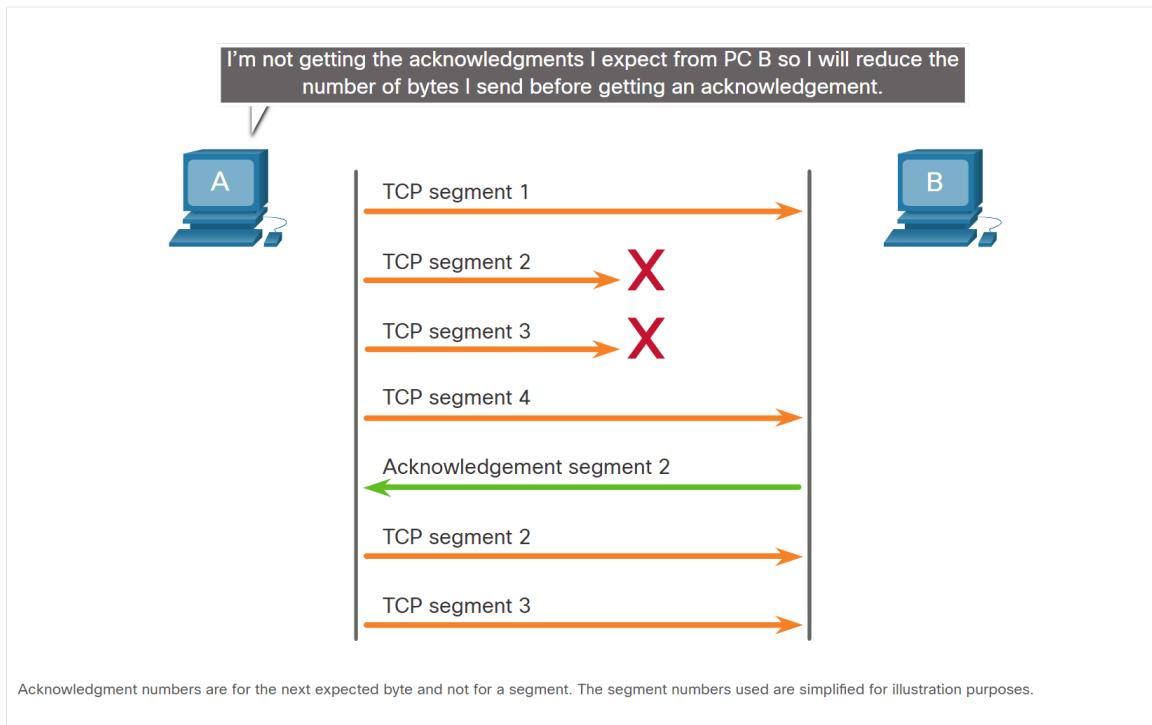
In all of these examples, I used very simple sequence numbers. In real situations, the sequence numbers get much larger and do not increase by 1 with each message. For the CCNA, just understand the concepts and don't worry about the exact numbers.



MSS= Maximum Segment Size



TCP Congestion Control



UDP

- User Datagram Protocol. Enables a process running on one host to send packets to a process running on another host. However, UDP does not confirm successful datagram transmission. (**datagram**) (Connectionless)
 - UDP does not provide reliability or flow control
 - it does not require an established connection
 - UDP is also known as a best-effort delivery protocol because there is no acknowledgment
 - Live video and voice applications can tolerate some data loss with minimal or no noticeable effect, and are perfectly suited to UDP.

UDP Header Fields



The table identifies and describes the four fields in a UDP header.

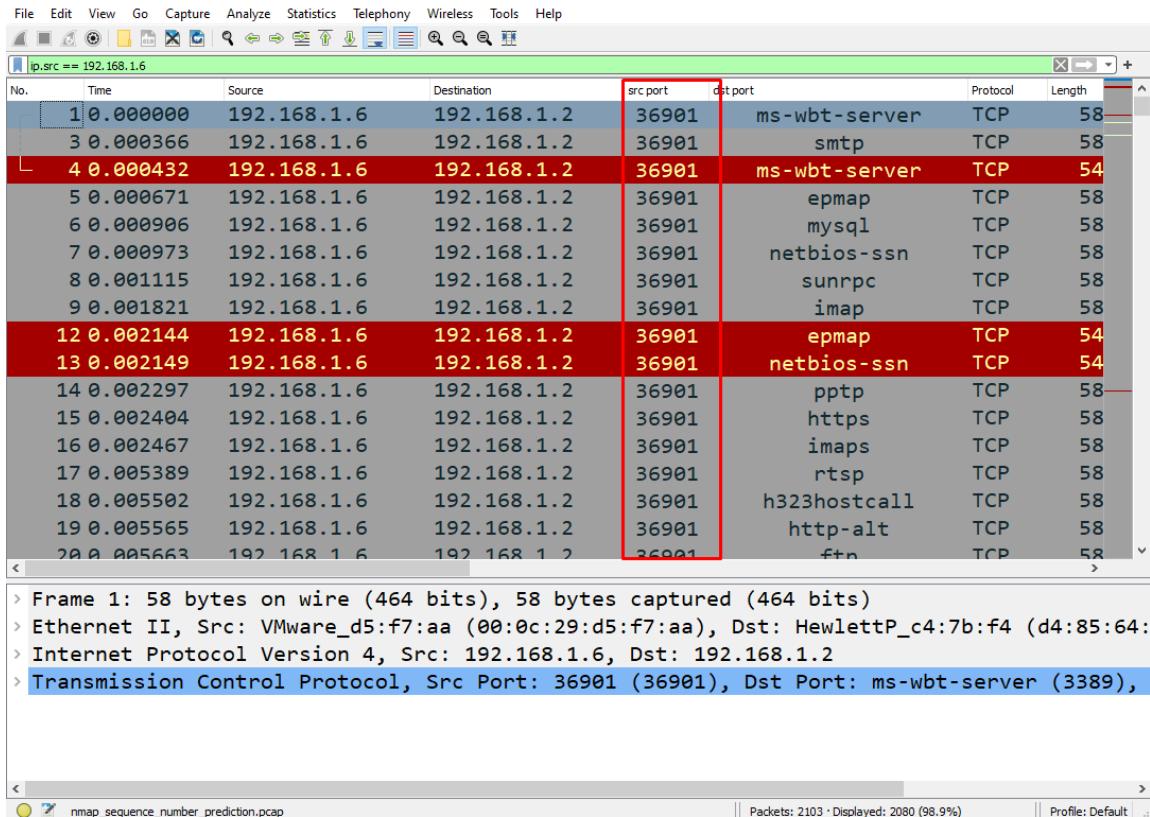
UDP Header Field	Description
Source Port	A 16-bit field used to identify the source application by port number.
Destination Port	A 16-bit field used to identify the destination application by port number.
Length	A 16-bit field that indicates the length of the UDP datagram header.
Checksum	A 16-bit field used for error checking of the datagram header and data.

Socket:

- The combination of the source IP address and source port number, or the destination IP address and destination port number is known as a socket.
- The socket is used to identify the server and service being requested by the client
- The socket on a web server might be **192.168.1.7:80**
- these two sockets combine to form a socket pair: 192.168.1.5:1099, 192.168.1.7:80
 - Sockets enable multiple processes, running on a client, to distinguish themselves from each other, and multiple connections to a server process to be distinguished from each other.

Suspicious TCP Traffic

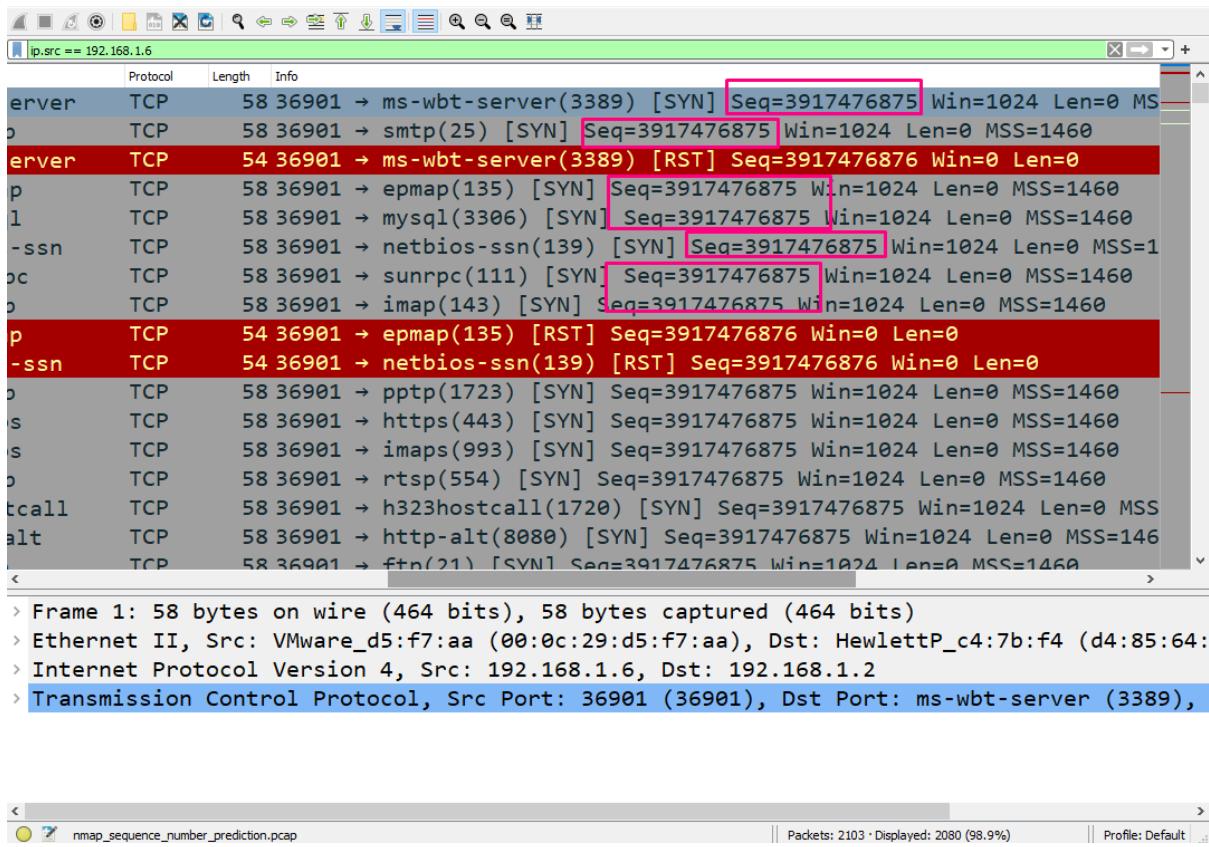
1. **Excessive SYN packets (scanning)**
2. **Usage of different flags**
3. **Single host to multiple ports or single host to multiple nodes (scanning)**
4. **Source Port Abnormalities :**
 - **Privileged (server) ports [1-1023]** ← Should remain unchanged during the entire connection
 - **Unprivileged (client)/ephemeral ports [1023-65535]** ← Chosen only for **one connection**. Can be chosen again after the connection closes.



- If you carefully look at packets , you will notice that the host 192.168.1.6 uses the same source port (36901) for multiple connection attempts to different ports of the remote host. This is abnormal.

5. Sequence Number Prediction & SYN Scanning

- One of the ways using which the venerable **Nmap** tool tries to perform **OS fingerprinting**, is by examining the Initial **Sequence Numbers (ISNs)** generated by the target host (after connections are being attempted to a listening port).
- Each TCP/IP stack (and subsequently each OS) features **its own way of generating Initial Sequence Numbers**.
- Nmap repeatedly used the **ISN** from the scanning host, while scanning **the different ports of the remote host**.
 - Unique ISNs should be used**, when attempting to connect to **different ports of a remote host**.



5. Destination Port Abnormalities

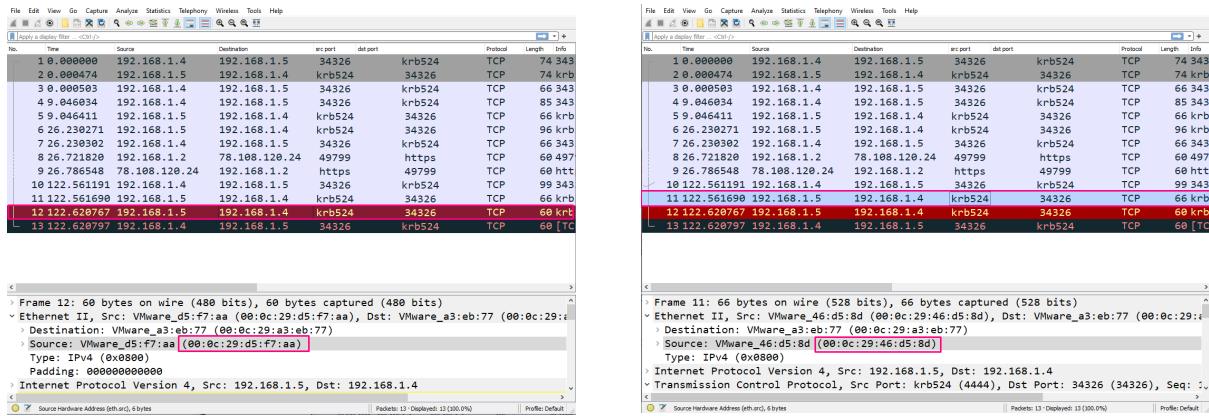
- when sending a **SYN request** to **Port 0** if the host responds with **RST, ACK** then it's **alive**
 - if not it will not respond

1 0.000000 192.168.1.6	192.168.1.4	TCP	60 2704 → 0 [SYN] Seq=0 Win=512 Len=0
2 0.000036 192.168.1.4	192.168.1.6	TCP	54 0 → 2704 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3 1.000539 192.168.1.6	192.168.1.4	TCP	60 2705 → 0 [SYN] Seq=0 Win=512 Len=0
4 1.000578 192.168.1.4	192.168.1.6	TCP	54 0 → 2705 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5 2.000999 192.168.1.6	192.168.1.4	TCP	60 2706 → 0 [SYN] Seq=0 Win=512 Len=0
6 2.001037 192.168.1.4	192.168.1.6	TCP	54 0 → 2706 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7 3.002394 192.168.1.6	192.168.1.4	TCP	60 2707 → 0 [SYN] Seq=0 Win=512 Len=0
8 3.002428 192.168.1.4	192.168.1.6	TCP	54 0 → 2707 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9 4.003789 192.168.1.6	192.168.1.4	TCP	60 2708 → 0 [SYN] Seq=0 Win=512 Len=0
10 4.003819 192.168.1.4	192.168.1.6	TCP	54 0 → 2708 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11 5.004594 192.168.1.6	192.168.1.4	TCP	60 2709 → 0 [SYN] Seq=0 Win=512 Len=0
12 5.004632 192.168.1.4	192.168.1.6	TCP	54 0 → 2709 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13 6.005871 192.168.1.6	192.168.1.4	TCP	60 2710 → 0 [SYN] Seq=0 Win=512 Len=0
14 6.005912 192.168.1.4	192.168.1.6	TCP	54 0 → 2710 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15 7.006765 192.168.1.6	192.168.1.4	TCP	60 2711 → 0 [SYN] Seq=0 Win=512 Len=0
16 7.006798 192.168.1.4	192.168.1.6	TCP	54 0 → 2711 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17 8.006788 192.168.1.6	192.168.1.4	TCP	60 2712 → 0 [SYN] Seq=0 Win=512 Len=0
18 8.007222 192.168.1.4	192.168.1.6	TCP	54 0 → 2712 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

6. TCP RST Attack

- an existing TCP connection can be cut apart through a crafted **TCP RST** packet sent either to the client or the server.
- For a successful RST attack to be executed, an attacker should have prior knowledge of the below.
 - Source & Destination Ports**

- Source & Destination IP
- “correct” Sequence Number



- the attacker spoofed the src ip and sent a TCP RST packet to the server to cut the connection.

7. TCP Session Hijacking

- TCP is **defenseless** against “packet injection” attacks.
- An attacker could choose to **hijack a whole TCP session**, instead of executing a simple TCP RST attack. (requires the same prior knowledge as the TCP RST attack.)

1 0.000000 192.168.1.4	192.168.1.5	TELNET	67 Telnet Data ...
2 0.001212 192.168.1.5	192.168.1.4	TELNET	67 Telnet Data ...
3 0.001829 192.168.1.4	192.168.1.5	TCP	66 36212 → 23 [ACK] Seq=3123452945 Ack=1235858240 Win=237 Len=0 TSval=4214429...
4 0.136173 192.168.1.4	192.168.1.5	TELNET	67 Telnet Data ...
5 0.136718 192.168.1.4	192.168.1.4	TELNET	67 Telnet Data ...
6 0.136958 192.168.1.4	192.168.1.5	TCP	66 36212 → 23 [ACK] Seq=3123452946 Ack=1235858241 Win=237 Len=0 TSval=4214429...
7 0.487999 192.168.1.4	192.168.1.5	TELNET	68 Telnet Data ...
8 0.488647 192.168.1.5	192.168.1.4	TELNET	68 Telnet Data ...
9 0.488844 192.168.1.4	192.168.1.5	TCP	66 36212 → 23 [ACK] Seq=3123452948 Ack=1235858243 Win=237 Len=0 TSval=4214429...
10 0.490814 192.168.1.5	192.168.1.4	TELNET	148 Telnet Data ...
11 0.490996 192.168.1.5	192.168.1.5	TCP	66 36212 → 23 [ACK] Seq=3123452948 Ack=1235858317 Win=237 Len=0 TSval=4214429...
12 6.264193 192.168.1.4	192.168.1.5	TELNET	67 Telnet Data ...
13 6.265435 192.168.1.4	192.168.1.4	TELNET	67 Telnet Data ...
14 6.265727 192.168.1.4	192.168.1.5	TCP	66 36212 → 23 [ACK] Seq=3123452949 Ack=1235858318 Win=237 Len=0 TSval=4214435...
15 6.282426 192.168.1.4	192.168.1.5	TCP	1078 [TCP Retransmission] 36212 → 23 PSH, ACK Seq=3123452948 Ack=1235858317
16 6.282994 192.168.1.5	192.168.1.4	TCP	78 [TCP ACKed unseen segment] 23 → 36212 [ACK] Seq=1235858318 Ack=312345397
17 14.488628 192.168.1.4	192.168.1.5	TELNET	63 Telnet Data ...
18 14.491266 192.168.1.5	192.168.1.4	TCP	66 23 → 36212 [ACK] Seq=1235858318 Ack=3123453981 Win=243 Len=0 TSval=2774555...
19 14.491272 192.168.1.5	192.168.1.4	TELNET	76 Telnet Data ...
20 14.696517 192.168.1.5	192.168.1.4	TELNET	238 Telnet Data ...
21 14.906519 192.168.1.5	192.168.1.4	TCP	240 [TCP Retransmission] 23 → 36212 PSH, ACK Seq=1235858318 Ack=3123453981 W...
22 15.322954 192.168.1.5	192.168.1.4	TCP	240 [TCP Retransmission] 23 → 36212 PSH, ACK Seq=1235858318 Ack=3123453981 W...
23 16.156205 192.168.1.5	192.168.1.4	TCP	240 [TCP Retransmission] 23 → 36212 PSH, ACK Seq=1235858318 Ack=3123453981 W...

- Telnet offers no encryption and thus, we can see every command the client issued and every result returned by the server in clear text.
- Let's analyze packet #15.
 - **TCP Retransmission** is displayed because the sequence number and the acknowledgement number of this packet are the same as the ones in packet #11.

- The MAC address of the client (192.168.1.4) in packet #15 (attacker) is different than the MAC address that is included in all previous packets related to this host.
- It looks like an attacker has taken over (hijacked) the whole Telnet session.
- This is also apparent in packet #17, that includes the MAC address of the attacker and the command the attacker issued (uname –a)
- The server has no defense mechanism to detect that the Telnet session is hijacked and sends the output of the uname –a command back to the 192.168.1.4 client.

```
.....uname -a
Linux kali 4.13.0-kali1-amd64 #1 SMP Debian 4.13.10-1kali2 (2017-11-08) x86_64 GNU/Linux
.]0;administrator@kali: ~.[01;31madministrator@kali.[00m:[01;34m~.[00m$
```

8. TCP Timestamps Option

the TCP Timestamps options can be abused in order to:

1. Determine the **patch level of a system**, through uptime analysis
2. Perform **host identification** using **clock skew**
3. Identify how a target **DMZ is structured**

<https://www.scip.ch/en/?labs.20150305>

9. Leveraging TCP Option Support & Ordering

- TCP/IP stacks, support a subset of the available TCP options and also, perform TCP option storing in their own unique order. Nmap leverages the above (and other things), in order to perform **OS fingerprinting**.

<https://nmap.org/nmap-fingerprinting-article.txt>

UDP-based attacks

DNS Command & Control, DNS exfiltration (to be discussed)

ICMP Abuse

1. ICMP Echo Request (8)

- To map live hosts
 - most sites nowadays disallow inbound and/or outbound ICMP echo requests.

2. ICMP Address Mask Request(17)/Reply(18)

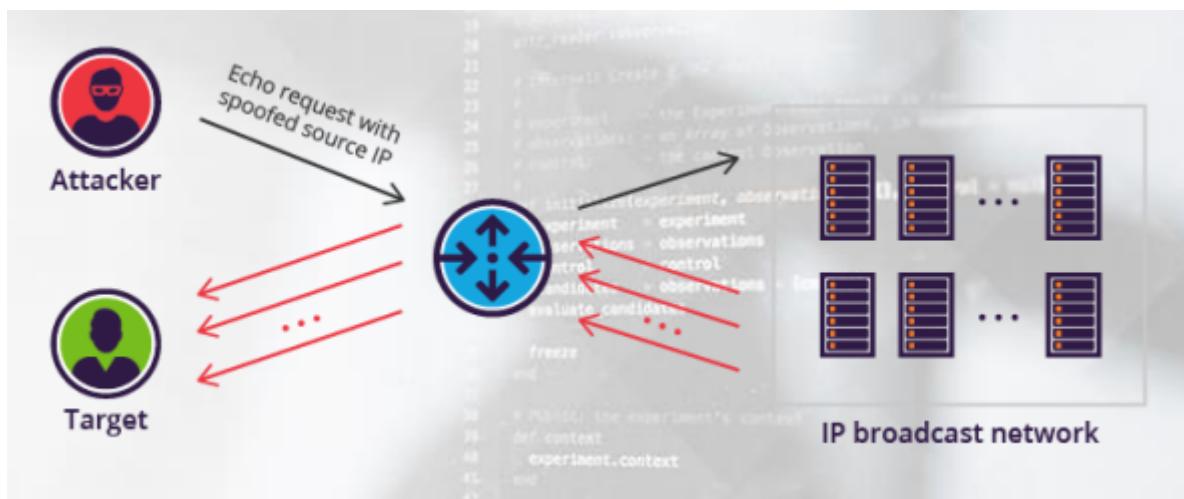
- can be used to identify a target host's subnet mask.

3. ICMP Timestamp Request(13)/Reply (14)

- can be used to obtain **timestamps** from remote systems.
 - used in zero days attack to see if the system got patched

4. Smurf Attack(DDoS)

- is executed as follows:
 1. the attacker sends fake echo requests to an intermediate ip broadcast network with a spoofed src ip of the target
 2. the requests is transmitted to all of the network hosts on the network
 3. all hosts on the network will send an ICMP echo reply to the target server causing it to get down



5. ICMP Tunneling

- ICMP can be misused to create a **covert channel of communication**. This can be achieved through ill-intended ICMP tunneling
 - Numerous ICMP tunneling solutions exists, but attackers seem to prefer the **ptunnel** one.

The screenshot shows a list of captured ICMP packets from a file named ptunnel.pcap. The traffic consists of ping requests and replies between two hosts. The sequence numbers for the echo requests and replies are highlighted with red boxes.

No.	Time	Source	Destination	src port	dst pc	Protocol	Length	Info	
1	0.000000	192.168.1.100	192.168.1.4			ICMP	70	Echo (ping) request	id=0x9a32, seq=0/0, tt...
2	0.000046	192.168.1.4	192.168.1.100			ICMP	70	Echo (ping) reply	id=0x9a32, seq=0/0, tt...
3	0.014989	192.168.1.4	192.168.1.100			ICMP	82	Echo (ping) reply	id=0x9a32, seq=0/0, tt...
4	0.015246	192.168.1.100	192.168.1.4			ICMP	82	Echo (ping) request	id=0x9a32, seq=1/256, ...
5	0.015262	192.168.1.4	192.168.1.100			ICMP	82	Echo (ping) reply	id=0x9a32, seq=1/256, ...
6	0.015319	192.168.1.4	192.168.1.100			ICMP	70	Echo (ping) reply	id=0x9a32, seq=1/256, ...
7	0.015439	192.168.1.4	192.168.1.100			ICMP	94	Echo (ping) reply	id=0x9a32, seq=2/512, ...
8	0.015612	192.168.1.100	192.168.1.4			ICMP	170	Echo (ping) request	id=0x9a32, seq=2/512, ...
9	0.015622	192.168.1.4	192.168.1.100			ICMP	170	Echo (ping) reply	id=0x9a32, seq=2/512, ...
10	0.017822	192.168.1.4	192.168.1.100			ICMP	86	Echo (ping) reply	id=0x9a32, seq=3/768, ...
11	0.017958	192.168.1.100	192.168.1.4			ICMP	94	Echo (ping) request	id=0x9a32, seq=3/768, ...
12	0.017977	192.168.1.4	192.168.1.100			ICMP	94	Echo (ping) reply	id=0x9a32, seq=3/768, ...
13	0.018171	192.168.1.4	192.168.1.100			ICMP	74	Echo (ping) reply	id=0x9a32, seq=4/1024, ...
14	0.018282	192.168.1.100	192.168.1.4			ICMP	74	Echo (ping) request	id=0x9a32, seq=4/1024, ...
15	0.018295	192.168.1.4	192.168.1.100			ICMP	74	Echo (ping) reply	id=0x9a32, seq=4/1024, ...
16	0.018440	192.168.1.4	192.168.1.100			ICMP	690	Echo (ping) reply	id=0x9a32, seq=5/1280, ...
17	1.0007995	192.168.1.100	192.168.1.4			ICMP	70	Echo (ping) request	id=0x9a32, seq=5/1280, ...
18	1.0008030	192.168.1.4	192.168.1.100			ICMP	70	Echo (ping) reply	id=0x9a32, seq=5/1280, ...
19	1.018125	192.168.1.4	192.168.1.100			ICMP	70	Echo (ping) reply	id=0x9a32, seq=6/1536, ...
20	1.729712	192.168.1.100	192.168.1.4			ICMP	72	Echo (ping) request	id=0x9a32, seq=6/1536, ...

Detection

- You can see **two or more replies** for each request
 - An Echo request and the associated Echo reply should have the **same length**. This isn't the case in this capture file
 - The payload size in some ICMP packets is a **lot bigger than normal**. see packet #16.
 - all ICMP packets of this capture file include the previously mentioned ptunnel magic value (**0xD5200880**), inside the ICMP payload.

- we can use `strings ptunnel.pcap` to identified the strings in the pcap

6. Abusing ICMP Redirect

- When two machines want to communicate with each other the router has to find the shortest path between them.
 - If there is an alternate **shorter** path between the two machines then, the router will send an **ICMP redirect packet** to the sender machine to change its routing

table so that it uses the shortest path.

- such ICMP redirect packets can be forged by an attacker and make the sender host redirect its packet to an **attacker-controlled** or non-existing destination.

```

1 0.000000 72:9b:2f:a0:90:... Broadcast ARP 60 Who has 10.100.13.126? Tell 10.100.13.20
2 0.000022 Vmware_a1:cb:34 72:9b:2f:a0:90:91 ARP 42 10.100.13.126 is at 00:50:56:a1:cb:34
3 0.251032 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
4 0.333645 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
5 0.397539 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
6 0.468985 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
7 0.555181 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
8 0.637849 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
9 0.722346 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
10 0.811009 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
11 0.900367 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
12 0.968061 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)
13 1.055575 10.100.13.1 10.100.13.126 ICMP 82 Redirect (Redirect for host)

Frame 3: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
Ethernet II, Src: Vmware_a1:cb:34 (00:50:56:a1:cb:34), Dst: 10.100.13.1 (08:00:27:00:00:01)
Internet Protocol Version 4, Src: 10.100.13.126 (10.100.13.126), Dst: 10.100.13.20 (10.100.13.20)
Internet Control Message Protocol
  Type: 5 (Redirect)
  Code: 1 (Redirect for host)
  Checksum: 0x3dfe [correct]
  [Checksum Status: Good]
  Gateway address: 10.100.13.20
  Internet Protocol Version 4, Src: 10.100.13.126, Dst: 10.23.56.100
  Transmission Control Protocol, Src Port: 55555, Dst Port: 80

```

- A large number of ICMP Redirect packets exist.
- In those packets the router instructs the client **10.100.13.126** to make a change in its routing table to use
- the gateway 10.100.13.20 for all subsequent packets. At this moment, you should check if the gateway 10.100.13.20 is a legitimate gateway.
- If you filter the whole capture file, based on the MAC address of the router (**10.100.13.1**) [eth.src==72:9b:2f:a0:90:91], you will notice that this MAC address is associated with the **10.100.13.20** machine.
- Even though it is not clearly visible in the capture file, every HTTP request can now be sniffed by the 10.100.13.20 host, as a result of the ICMP Redirect attack.

Application Layer

Network Basic Input/Output System NetBIOS

- a set of protocols developed in for Windows only in order to provide services for the session layer

NetBIOS provides three services :

- (NBNS) Name service (works over **UDP port 137**) for **name registration** and **name to IP** address resolution.

- a. it was later replaced by **DNS**
- 2. **(NBDS) Datagram** distribution service (works over **UDP** port **138**) for **service announcements** by clients and servers.
- 3. **(NBSS) Session** service (works over **TCP** port **139**) for **session negotiation between hosts**. This is used for **accessing files, opening directories**, and so on.
- There are additional protocols such as Server Message Block (**SMB**)

SMB

- a protocol that is used for browsing directories, copying files, accessing services such as printers, and several other operations over the network
- SMB runs on top of the **session layer** protocols such as **NetBIOS** as originally designed
 - can also run directly over **TCP** port **445**
- Common Internet File System (**CIFS**) is a form, or flavor, of SMB.

Detection

- **SMB works in a client-server model**
- Code **0** means **STATUS_OK**, which implies that everything works fine and there is no problem. Any other code should be examined.

203.12.106.10	SMB	93	NT Trans Response, FID: 0x0001, NT NOTIFY, Error: STATUS_CANCELLED
203.12.106.10	SMB	478	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PROCESSING_REQUIRED
203.12.106.10	SMB	93	Tree Connect AndX Response, Error: STATUS_ACCESS_DENIED
203.12.106.10	SMB	478	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PROCESSING_REQUIRED
203.12.106.10	SMB	93	Tree Connect AndX Response, Error: STATUS_ACCESS_DENIED
203.12.106.10	SMB	478	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PROCESSING_REQUIRED
203.12.106.10	SMB	93	Tree Connect AndX Response, Error: STATUS_ACCESS_DENIED
203.12.106.14	SMB	93	Trans2 Response, GET_DFS_REFERRAL, Error: STATUS_NO_SUCH_DEVICE
203.12.106.14	SMB	93	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED

- **NULL session** : anonymous and passwordless **authentication**, if allowed it could be used to execute various **RPC** calls and subsequently perform information gathering or user enumeration.

TCP	57500 → 445 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=969964135 TSecr=1641308
TCP	445 → 57500 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=969964135 TSecr=1641308
TCP	57500 → 445 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=969964135 TSecr=1641308
SMB	Negotiate Protocol Request
SMB	Negotiate Protocol Response
TCP	57500 → 445 [ACK] Seq=54 Ack=118 Win=20212 Len=0 TSval=969964135 TSecr=1641308
SMB	Session Setup AndX Request, User: anonymous
SMB	Session Setup AndX Response
SMB	Tree Connect AndX Request, Path: \\192.168.57.105\IPC\$
SMB	Tree Connect AndX Response
SMB	NT Create AndX Request, FID: 0x4000, Path: \samr
SMB	NT Create AndX Response, FID: 0x4000
DCERPC	Bind: call_id: 1094795585, Fragment: Single, 1 context items: SAMR V1.0 (32bit NDR)
SMB	Write AndX Response, FID: 0x4000, 72 bytes
SMB	Read AndX Request, FID: 0x4000, 2048 bytes at offset 0
DCERPC	Bind_ack: call_id: 1094795585, Fragment: Single, max_xmit: 2048 max_recv: 2048, 1
SAMR	Connect4 request
SMB	Write AndX Response, FID: 0x4000, 84 bytes
SMB	Read AndX Request, FID: 0x4000, 4097 bytes at offset 0
SAMR	Connect4 response
SAMR	EnumDomains request
SMB	Write AndX Response, FID: 0x4000, 52 bytes
SMB	Read AndX Request, FID: 0x4000, 4097 bytes at offset 0
SAMR	EnumDomains response
SAMR	LookupDomain request, User-PC[Long frame (2 bytes)]
SMB	Write AndX Response, FID: 0x4000, 80 bytes
SMB	Read AndX Request, FID: 0x4000, 4097 bytes at offset 0
SAMR	LookupDomain response
SAMR	OpenDomain request
SMB	Write AndX Response, FID: 0x4000, 76 bytes
SMB	Read AndX Request, FID: 0x4000, 4097 bytes at offset 0
SAMR	OpenDomain response
SAMR	QueryDisplayInfo request
SMB	Write AndX Response, FID: 0x4000, 60 bytes
SMB	Read AndX Request, FID: 0x4000, 4097 bytes at offset 0
SAMR	QueryDisplayInfo response
SAMR	Close request
SMB	Write AndX Response, FID: 0x4000, 44 bytes
SMB	Read AndX Request, FID: 0x4000, 4097 bytes at offset 0
SAMR	Close response
SAMR	Close request
SMB	Write AndX Response, FID: 0x4000, 44 bytes
SMB	Read AndX Request, FID: 0x4000, 4097 bytes at offset 0
SAMR	Close response
SMB	Tree Disconnect Request
SMB	Tree Disconnect Response
SMB	Logoff AndX Request
SMB	Logoff AndX Response
TCP	57500 → 445 [FIN, ACK] Seq=1943 Ack=2587 Win=35328 Len=0 TSval=969964141 TSecr=1641308

MSRPC

- RPC (Remote Procedure Call) mechanism allows an application to seamlessly invoke remote procedures, as if these procedures were executed locally
- MSRPC is the Microsoft implementation of the **DCE RPC** mechanism.
- **File operations** utilize **SMB/CIFS**, whereas **administrative operations, resource management** operations etc. utilize **MSRPC**.

There are multiple MSRPC implementations:

- RPC over **SMB**
- DCOM (RPC directly over TCP/UDP) [TCP/UDP port 135]

TCP Hand Shake

No.	Time	Source	Destination	srcport	dstport	Protocol	Length	Info
129	13.162956	192.168.52.61	192.168.52.10	49168	135	TCP		66 49168 → 135 [SYN] Seq=1270918437 Win=8192
130	13.163090	192.168.52.10	192.168.52.61	135	49168	TCP		66 135 → 49168 [SYN, ACK] Seq=582498090 Ack=582498091
131	13.163230	192.168.52.61	192.168.52.10	49168	135	TCP		54 49168 → 135 [ACK] Seq=1270918438 Ack=582498091
132	13.165417	192.168.52.61	192.168.52.10	49168	135	DCERPC		214 Bind: call_id: 2, Fragment: Single, 3 controls
133	13.165587	192.168.52.10	192.168.52.61	135	49168	DCERPC		162 Bind_ack: call_id: 2, Fragment: Single, 3 controls
134	13.165875	192.168.52.61	192.168.52.10	49168	135	EPM		210 Map request, DRSSUAPI, 32bit NDR
135	13.166136	192.168.52.10	192.168.52.61	135	49168	EPM		206 Map response, DRSSUAPI, 32bit NDR
167	13.369138	192.168.52.61	192.168.52.10	49168	135	TCP		54 49168 → 135 [ACK] Seq=1270918754 Ack=582498091
195	16.136993	192.168.52.61	192.168.52.10	49168	135	EPM		210 Map request, DRSSUAPI, 32bit NDR
196	16.137415	192.168.52.10	192.168.52.61	135	49168	EPM		206 Map response, DRSSUAPI, 32bit NDR
230	16.333563	192.168.52.61	192.168.52.10	49168	135	TCP		54 49168 → 135 [ACK] Seq=1270918910 Ack=582498091

```

> Frame 130: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: RealtekU_0a:70:f7 (52:54:00:0a:70:f7), Dst: RealtekU_c6:63:ca (52:54:00:c6:63:ca)
> Internet Protocol Version 4, Src: 192.168.52.10, Dst: 192.168.52.61
> Transmission Control Protocol, Src Port: epmap (135), Dst Port: 49168 (49168), Seq: 582498090, Ack: 1270918438, Len: 0

```

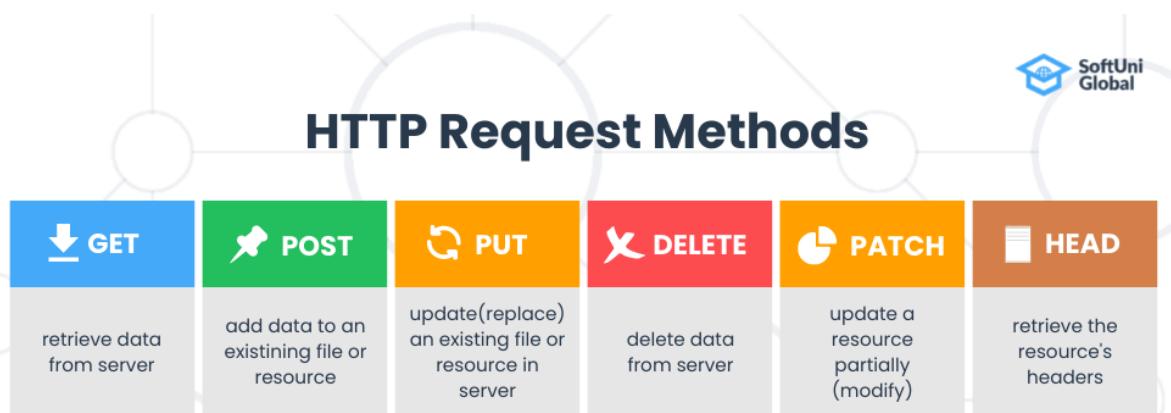
schannel_win7-member_to_win2k3-dc.pcap

Packets: 611 · Displayed: 11 (1.8%) · Profile: Default

- RPC over **HTTP or HTTPS** [TCP/UDP port 593]

HTTP

- HTTP uses methods to perform various operations.
 - Not all methods will be permitted by web server.



- CONNECT** is used to open a two-way socket connection to the remote server;
- OPTIONS** is used to describe the communication options for specified resource;
- TRACE** is designed for diagnostic purposes during the development.
- HEAD** retrieves the resource's headers, without the resource itself.

Normal vs Suspicious HTTP Trafic

Normal HTTP Traffic	Suspicious HTTP Traffic
Port 80, TCP Port 8080, TCP (used as alternate) Port 8088, TCP (used as alternate)	Malicious binaries (backdoors), scripts, <u>web shells</u> , etc. will use this port because typically in all corporate environments the port is open.
<u>Plaintext</u> traffic	If the traffic is <u>encrypted</u> then most likely it's being used for malicious traffic. Malicious traffic can be in plaintext as well.
Web server typically in FQDN format.	Server will point to an <u>IP address</u> instead of FQDN format.

Normal HTTP Traffic example:

Time	Source	Destination	Protocol	Length	Info
3 0.0508820...	10.54.15...	10.54.15...	TCP	74	50982 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK
5 0.0965761...	10.54.15...	10.54.15...	TCP	74	80 → 50982 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS
6 0.0966111...	10.54.15...	10.54.15...	TCP	66	50982 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=25
7 0.0968141...	10.54.15...	10.54.15...	HTTP	347	GET / HTTP/1.1
8 0.1469519...	10.54.15...	10.54.15...	TCP	66	80 → 50982 [ACK] Seq=1 Ack=282 Win=15552 Len=0 TSval=654
9 0.1809993...	10.54.15...	10.54.15...	HTTP	200	OK (text/html)

- We are seeing 6 packets (4 relating to TCP and 2 relating to HTTP).
 - Packets **3-6** is the **TCP Handshake**. HTTP relies on TCP for reliability.
 - Packet **7** we notice a **HTTP method (GET)**.
 - Packet **9** we notice a **HTTP response code (200 OK)**.
 - port **80** is used
- we can see the content of the HTTP Stream with **Follow -> Follow TCP Stream**

Wireshark - Follow TCP Stream (tcp.stream eq 2) · basic-http-traffic

```

POST /login.php HTTP/1.1
Host: 10.54.15.68
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.54.15.68/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

user=elsstudent&pass=testpassword&login=loginHTTP/1.1 200 OK
Date: Tue, 23 May 2017 17:23:56 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u14
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 265
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

.....mQ=o. ....qawP....".C..#.k@.&.ku...;.pm...[wz...#.....=8j.Z...t.(j;C.."iON..
$....aK:.P.....,w...,H.g.T.. .2..C$...d&g>,rc....LB..P.F..?.4.G..D.m.....
nA ..6'..11..7'h..1o2.....* $U..SzJllyv.R.....3.....\dL = chd ..!
3 client pkts, 3 server pkts, 3 turns.

Entire conversation (973 bytes) Show and save data as ASCII Stream 2
Find: Help Filter Out This Stream Print Save as... Back Close

```

- or **Select Statistics -> Conversations** Under the **TCP** tab in Conversations we can see there are 3 TCP Streams. From here we can select a stream and choose Follow Stream from bottom right corner.

Ethernet - 4	IPv4 - 1	IPv6 - 1	TCP - 3	UDP									
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.54.15.100	50982	10.54.15.68	80	13	3198	7	1030	6	2168	0.050883	5.2850	1559	
10.54.15.100	51008	10.54.15.68	80	10	1470	5	630	5	840	20.862049	5.1468	979	
10.54.15.100	51012	10.54.15.68	80	10	1649	5	773	5	876	42.917399	5.1707	1195	

Observations control

Conversation Types
Copy
Follow Stream...
Graph...
Close

Malicious HTTP Traffic Example

- The attacker is attempting sql injection manually

Source	Destination	Protocol	Length	Info
10.124.211.200	10.124.211.96	HTTP	373	GET /newsdetails.php?id=26%27 HTTP/1.1
10.124.211.96	10.124.211.200	TCP	1391	[TCP segment of a reassembled PDU]
10.124.211.200	10.124.211.96	TCP	68	33020 → 80 [ACK] Seq=1258 Ack=6640 Win=44800 Len=0 TSval=127992 TSecr=224575
10.124.211.96	10.124.211.200	HTTP	68	HTTP/1.1 200 OK (text/html)
10.124.211.200	10.124.211.96	TCP	68	33020 → 80 [ACK] Seq=1258 Ack=6642 Win=44800 Len=0 TSval=127992 TSecr=224575
10.124.211.96	10.124.211.200	TCP	68	80 → 33020 [FIN, ACK] Seq=6644 Ack=1258 Win=18768 Len=0 TSval=225826 TSecr=127992
10.124.211.200	10.124.211.96	TCP	68	33020 → 80 [FIN, ACK] Seq=1256 Ack=6643 Win=44800 Len=0 TSval=129243 TSecr=225826
10.124.211.96	10.124.211.200	TCP	68	80 → 33020 [ACK] Seq=6643 Ack=1259 Win=18768 Len=0 TSval=225842 TSecr=129243
10.124.211.200	10.124.211.96	TCP	74	33022 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=129973 TSecr=0
10.124.211.96	10.124.211.200	TCP	74	80 → 33022 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_PERM=1 TSval=226572
10.124.211.200	10.124.211.96	HTTP	389	GET /newsdetails.php?id=26%20and%201=1--%20- HTTP/1.1
10.124.211.96	10.124.211.200	TCP	68	80 → 33022 [ACK] Seq=1 Ack=324 Win=16552 Len=0 TSval=226590 TSecr=129989
10.124.211.96	10.124.211.200	TCP	84	[TCP Previous segment not captured] [TCP segment of a reassembled PDU]
10.124.211.200	10.124.211.96	TCP	78	[TCP Window Update] 33022 → 80 [ACK] Seq=324 Ack=1 Win=0 TSval=130036 Len=0 TSval=130008 TSecr=130008
10.124.211.96	10.124.211.200	TCP	1391	[TCP Out-Of-Order] 80 → 33022 [ACK] Seq=1 Ack=324 Win=15552 Len=1325 TSval=226591
10.124.211.200	10.124.211.96	TCP	68	33022 → 80 [ACK] Seq=324 Ack=1344 Win=33280 Len=0 TSval=130008 TSecr=226591
10.124.211.96	10.124.211.200	TCP	68	80 → 33022 [FIN, ACK] Seq=1344 Ack=324 Win=131258 TSval=131258 TSecr=226591
10.124.211.200	10.124.211.96	TCP	68	33022 → 80 [ACK] Seq=325 Ack=1345 Win=33280 Len=0 TSval=131259 TSecr=227842
10.124.211.96	10.124.211.200	TCP	68	80 → 33022 [ACK] Seq=1345 Ack=325 Win=15552 Len=0 TSval=227852 TSecr=131258
10.124.211.200	10.124.211.96	TCP	74	33024 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=131888 TSecr=0
10.124.211.96	10.124.211.200	TCP	74	80 → 33024 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_PERM=1 TSval=228481 TSecr=0
10.124.211.200	10.124.211.96	HTTP	389	GET /newsdetails.php?id=26%20and%201=2--%20- HTTP/1.1
10.124.211.96	10.124.211.200	TCP	68	80 → 33024 [ACK] Seq=1 Ack=321 Win=15552 Len=0 TSval=131899 TSecr=228492 TSecr=131899
10.124.211.200	10.124.211.96	HTTP	1285	HTTP/1.1 200 OK (text/html)
10.124.211.200	10.124.211.96	TCP	66	33024 → 80 [ACK] Seq=324 Ack=1220 Win=32128 Len=0 TSval=131910 TSecr=228493
10.124.211.200	10.124.211.96	TCP	66	33024 → 80 [FIN, ACK] Seq=324 Ack=1220 Win=32128 Len=0 TSval=133160 TSecr=228493

- By further inspection in packet #20, we see that the User-Agent is Firefox and the OS is Linux

```
► Frame 20: 373 bytes on wire (2984 bits), 373 bytes captured (2984 bits)
► Ethernet II, Src: 1a:3a:46:b7:43:91 (1a:3a:46:b7:43:91), Dst: VMware_a1:4e:f0 (00:50:56:a1:4e:f0)
► Internet Protocol Version 4, Src: 10.124.211.200, Dst: 10.124.211.96
► Transmission Control Protocol, Src Port: 33020, Dst Port: 80, Seq: 951, Ack: 5315, Len: 307
▼ Hypertext Transfer Protocol
  ► GET /newsdetails.php?id=26%27 HTTP/1.1\r\n
    Host: 10.124.211.96\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://10.124.211.96/newsdetails.php?id=26%27]
  [HTTP request 4/4]
  [Prev request in frame: 15]
  [Response in frame: 23]
```

- Strangely, packet #56, and the packet after that, packet #73, don't seem to contain any SQL injection queries. Maybe he quit?

Time	Source	Destination	Protocol	Length	Info
56.34.168059	10.124.211.200	10.124.211.96	HTTP	266	GET /newsdetails.php?id=1 HTTP/1.1
56.34.168092	10.124.211.200	10.124.211.96	TCP	1391	[TCP segment of a reassembled PDU]
59.34.169364	10.124.211.200	10.124.211.96	TCP	68	33028 → 80 [ACK] Seq=1 Ack=1 Win=14460 Len=0 MSS=1637 SACK_PERM=1 TSval=230739 TSecr=134145 WS=4
59.34.180457	10.124.211.200	10.124.211.96	TCP	74	33030 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=134158 TSecr=0 WS=128
60.34.202983	10.124.211.96	10.124.211.200	TCP	68	80 → 33026 [ACK] Seq=0 Ack=1 Win=15552 Len=0 TSval=230747 TSecr=134154
61.34.206414	10.124.211.96	10.124.211.200	TCP	1391	[TCP segment of a reassembled PDU]
62.34.206432	10.124.211.96	10.124.211.200	TCP	82	33028 → 80 [ACK] Seq=204 Ack=1236 Win=32128 Len=0 TSval=134165 TSecr=230748
63.34.206495	10.124.211.96	10.124.211.200	HTTP	82	HTTP/1.1 200 OK (text/html)
64.34.206501	10.124.211.96	10.124.211.200	TCP	66	33028 → 80 [ACK] Seq=204 Ack=1242 Win=32128 Len=0 TSval=134165 TSecr=230748
65.34.206509	10.124.211.96	10.124.211.200	TCP	68	80 → 33028 [FIN, ACK] Seq=1342 Ack=201 Win=15552 Len=0 TSval=230749 TSecr=134154
66.34.207131	10.124.211.200	10.124.211.96	TCP	66	33026 → 80 [FIN, ACK] Seq=201 Ack=1343 Win=32128 Len=0 TSval=134165 TSecr=230749
67.34.218945	10.124.211.96	10.124.211.200	TCP	74	80 → 33030 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_PERM=1 TSval=230751 TSecr=134158 WS=4
68.34.218972	10.124.211.200	10.124.211.96	TCP	66	33030 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=134168 TSecr=230751
69.34.230022	10.124.211.200	10.124.211.96	TCP	74	33032 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=134171 TSecr=0 WS=128
70.34.248217	10.124.211.96	10.124.211.200	TCP	68	80 → 33026 [ACK] Seq=1343 Ack=202 Win=15552 Len=0 TSval=230758 TSecr=134165
71.34.265096	10.124.211.96	10.124.211.200	TCP	74	80 → 33028 [ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_PERM=1 TSval=230764 TSecr=134171 WS=4
72.34.269934	10.124.211.96	10.124.211.200	TCP	66	33026 → 80 [ACK] Seq=0 Ack=1 Win=14480 Len=0 TSval=134161 TSecr=230764
73.35.128264	10.124.211.200	10.124.211.96	HTTP	266	GET /newsdetails.php?id=1 HTTP/1.1
74.35.128569	10.124.211.200	10.124.211.96	TCP	74	33034 → 80 [SYN] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=134395 TSecr=0 WS=128
75.35.166226	10.124.211.96	10.124.211.200	TCP	66	80 → 33028 [ACK] Seq=1 Ack=201 Win=15552 Len=0 TSval=230988 TSecr=134395
76.35.166267	10.124.211.96	10.124.211.200	TCP	74	80 → 33034 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_PERM=1 TSval=230988 TSecr=134395 WS=4
77.35.166341	10.124.211.200	10.124.211.96	TCP	66	33034 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=134405 TSecr=230988
78.35.168806	10.124.211.96	10.124.211.200	TCP	1391	[TCP segment of a reassembled PDU]
79.35.168808	10.124.211.200	10.124.211.96	TCP	68	33028 → 80 [ACK] Seq=201 Ack=1342 Win=32128 Len=0 TSval=134405 TSecr=230988
80.35.168926	10.124.211.96	10.124.211.200	HTTP	68	80 → 33028 [FIN, ACK] Seq=201 Ack=1342 Win=32128 Len=0 TSval=134405 TSecr=230988
81.35.168943	10.124.211.200	10.124.211.96	TCP	66	33028 → 80 [ACK] Seq=201 Ack=1342 Win=15552 Len=0 TSval=230988 TSecr=134395
82.35.168962	10.124.211.96	10.124.211.200	TCP	66	80 → 33028 [FIN, ACK] Seq=201 Ack=1343 Win=32128 Len=0 TSval=134406 TSecr=230988
83.35.170154	10.124.211.200	10.124.211.96	TCP	66	33028 → 80 [FIN, ACK] Seq=201 Ack=1343 Win=16552 Len=0 TSval=134406 TSecr=230988

- taking a closer look we found out that The User-Agent for this HTTP GET Request is Sqlmap. So the attacker didn't quit, he escalated.

```

▶ Frame 56: 266 bytes on wire (2128 bits), 266 bytes captured (2128 bits)
▶ Ethernet II, Src: 1a:3a:46:bf:43:91 (1a:3a:46:bf:43:91), Dst: VMware_a1:4e:f0 (00:50:56:a1:4e:f0)
▶ Internet Protocol Version 4, Src: 10.124.211.200, Dst: 10.124.211.96
▶ Transmission Control Protocol, Src Port: 33025, Dst Port: 80, Seq: 1, Ack: 1, Len: 200
▼ Hypertext Transfer Protocol
  ▶ GET /newsdetails.php?id=1 HTTP/1.1\r\n
    Accept-Encoding: gzip,deflate\r\n
    Host: 10.124.211.96\r\n
    Accept: */*\r\n
    User-Agent: sqlmap/1.1.4#stable (http://sqlmap.org)\r\n
    Connection: close\r\n
    Cache-Control: no-cache\r\n
  \r\n
  [Full request URI: http://10.124.211.96/newsdetails.php?id=1]
  [HTTP request 1/1]
  [Response in frame: 63]

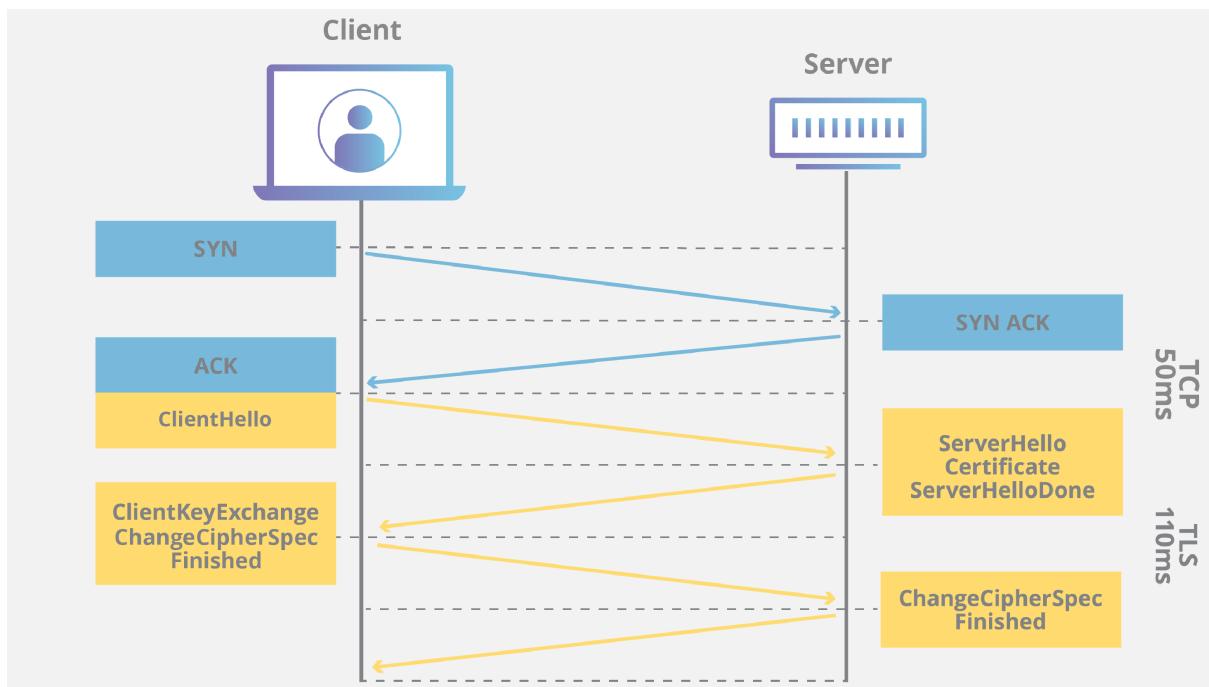
```

HTTPS

- HTTPS also establishes a **handshake** similar to TCP but more **complicated**.

Below is a brief summary:

- Both the client and the server need to agree on the **protocol version**.
- Both the client and the server need to select **cryptographic algorithms**.
- **Optionally authenticate** to each other.
- Use **public key encryption techniques** to establish secure communications.



Normal vs Suspicious HTTPS Traffic

Normal HTTPS Traffic	Suspicious HTTPS Traffic
Port 443, TCP Port 8443, TCP (used as alternate)	Malicious binaries (backdoors), scripts, web shells, etc. will use this port because typically in all corporate environments the port is open.
Encrypted traffic	If the traffic is not encrypted and Secure Sockets Layer packet details are empty within packet details then that will fall under suspicious.
Web server typically in FQDN format.	Server will point to an IP address instead of FQDN format.

Normal HTTPS Trafic

- the Secure Sockets Layer portion of the packet details should not be empty
- in **Client Hello packet**. We see the following:
 - Content Type = Handshak
 - Handshake Protocol: Client Hello
 - Version: TLS 1.2
 - Cipher Suites (11 suites)
 - Compression Method (1 method)

```

▶ Frame 25: 233 bytes on wire (1864 bits), 233 bytes captured (1864 bits)
▶ Ethernet II, Src: 26:11:59:88:53:02 (26:11:59:88:53:02), Dst: VMware_a1:61:66 (00:50:56:a1:61:66)
▶ Internet Protocol Version 4, Src: 10.54.15.100, Dst: 10.54.15.15
▶ Transmission Control Protocol, Src Port: 45114, Dst Port: 443, Seq: 1, Ack: 1, Len: 167
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22) ←
    Version: TLS 1.0 (0x0301)
    Length: 162
  ▼ Handshake Protocol: Client Hello ←
    Handshake Type: Client Hello (1)
    Length: 158
    Version: TLS 1.2 (0x0303) ←
    ▶ Random
      Session ID Length: 0
      Cipher Suites Length: 22 ←
    ▼ Cipher Suites (11 suites)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b) ←
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
      Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
    Compression Methods Length: 1 ←
  ▼ Compression Methods (1 method)
    Compression Method: null (0)
    Extensions Length: 95
    ▶ Extension: renegotiation_info
    ▶ Extension: elliptic_curves
    ▶ Extension: ec_point_formats
    ▶ Extension: SessionTicket TLS
    ▶ Extension: next_protocol_negotiation
    ▶ Extension: Application Layer Protocol Negotiation
    ▶ Extension: status_request
    ▶ Extension: signature_algorithms

```

- in the server's response, Server Hello packet

```

▶ Frame 27: 1391 bytes on wire (11128 bits), 1391 bytes captured (11128 bits)
▶ Ethernet II, Src: VMware_a1:f4:d0 (00:50:56:a1:f4:d0), Dst: 26:11:59:88:53:02 (26:11:59:88:53:02)
▶ Internet Protocol Version 4, Src: 10.54.15.15, Dst: 10.54.15.100
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 45114, Seq: 1, Ack: 168, Len: 1325
└ Secure Sockets Layer
  └ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 61
    └ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 57
      Version: TLS 1.2 (0x0303)
    └ Random
      GMT Unix Time: May 23, 2017 13:27:38.000000000 EDT ←
      Random Bytes: 2000d7125ade0022e9441d5121c77b5e3cb88e6b5fd2242e...
      Session ID Length: 0 ←
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ←
      Compression Method: null (0)
      Extensions Length: 17
    ▶ Extension: renegotiation_info
    ▶ Extension: ec_point_formats
    ▶ Extension: SessionTicket TLS
  └ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 1011
    └ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 1007
      Certificates Length: 1004 ←
    ▶ Certificates (1004 bytes)
      Certificate Length: 1001 ←
    ▶ Certificate: 308203e5308202cda003020102020900d98303cf87501375... (pkcs-9-at-emailAddress=)
      └ signedCertificate
        version: v3 (2)
        serialNumber: -2773368755666807947
        ▶ signature (sha1WithRSAEncryption)
        ▶ issuer: rdnSequence (0)
        ▶ validity
        ▶ subject: rdnSequence (0)
        ▶ subjectPublicKeyInfo
        ▶ extensions: 3 items
        ▶ algorithmIdentifier (sha1WithRSAEncryption)
        Padding: 0
        encrypted: 2326c8138a9c0a09ff804ee8e6909cae6f34ae00cf343ae9...

```

- Here we see the **Server Key Exchange** which will be followed by the **Client Key Exchange** packet. (step #3 in the establishment of an SSL/TLS session)

```

Frame 29: 170 bytes on wire (1360 bits), 170 bytes captured (1360 bits)
Ethernet II, Src: VMware_a1:f4:d0 (00:50:56:a1:f4:d0), Dst: 26:11:59:88:53:02 (26:11:59:88:53:02)
Internet Protocol Version 4, Src: 10.54.15.15, Dst: 10.54.15.100
Transmission Control Protocol, Src Port: 443, Dst Port: 45114, Seq: 1326, Ack: 168, Len: 104
[2 Reassembled TCP Segments (338 bytes): #27(243), #29(95)]
Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 333
    ▾ Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 329
      ▾ EC Diffie-Hellman Server Params
        Curve Type: named_curve (0x03)
        Named Curve: secp256r1 (0x0017)
        Pubkey Length: 65
        Pubkey: 04401daa41bf0a036acffe3ce86c112c109af374b2ef1326...
        ▾ Signature Hash Algorithm: 0x0401
          Signature Hash
          Signature Hash
          Signature Length:
          Signature: 8f3e65...
  ▾ Secure Sockets Layer
    ▾ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 4
      ▾ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 66
        ▾ EC Diffie-Hellman Client Params
          Pubkey Length: 65
          Pubkey: 04496c4e42312aa0f1b9855834438ee5d7f97745533bfc5e...
    ▾ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
    ▾ TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 40
      ▾ Handshake Protocol: Hello Request
        Handshake Type: Hello Request (0)
        Length: 0
      ▾ Handshake Protocol: Hello Request
        Handshake Type: Hello Request (0)
        Length: 0

```

- This is the **last packet** and the **handshake** between the server and client is now **complete**.

```

▶ Frame 34: 324 bytes on wire (2592 bits), 324 bytes captured (2592 bits)
▶ Ethernet II, Src: Vmware_a1:f4:d0 (00:50:56:a1:f4:d0), Dst: 26:11:59:88:53:02 (26:11:59:88:53:02)
▶ Internet Protocol Version 4, Src: 10.54.15.15, Dst: 10.54.15.100
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 45114, Seq: 1430, Ack: 604, Len: 258
▼ Secure Sockets Layer
  ▶ TLSv1.2 Record Layer: Handshake Protocol: New Session Ticket ←
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 202
  ▶ Handshake Protocol: New Session Ticket
    Handshake Type: New Session Ticket (4)
    Length: 198
    ▶ TLS Session Ticket
      Session Ticket Lifetime Hint: 300
      Session Ticket Length: 192
      Session Ticket: c87ec842e1a7e7c2fd503729435f618d50f9e59487ae8647...
  ▶ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message ←
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 40
    Handshake Protocol: Encrypted Handshake Message

```

- The rest of the packets between these two devices will now be **encrypted**.
- The traffic is unreadable, but if this is internal traffic within our corporate environment, then, it is feasible to decrypt this traffic using the **private key** from the internal server.

```

▶ Frame 35: 770 bytes on wire (6160 bits), 770 bytes captured (6160 bits)
▶ Ethernet II, Src: Vmware_a1:f4:d0 (00:50:56:a1:f4:d0), Dst: 26:11:59:88:53:02 (26:11:59:88:53:02)
▶ Internet Protocol Version 4, Src: 10.54.15.15, Dst: 10.54.15.100
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 45114, Seq: 1688, Ack: 604, Len: 704
▼ Secure Sockets Layer
  ▶ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 32
    Encrypted Application Data: bb006752c8e53aef6bb13c15ff590c829883685da8b96e44...

```

Malicious HTTPS Traffic Example

- we can use this filter `ssl.record.content_type == 22` in order to get the **SSL/TLS handshakes**

No.	Time	Source	Destination	src port	dst port	Protocol	Length	Info
1	0.000000	215.255.186.158	251.217.119.1...	55726	https	TLSv1	364	Client Hello
2	0.040015	215.255.186.158	251.217.119.1...	55731	https	TLSv1	364	Client Hello
14	0.543463	215.255.186.158	251.217.119.1...	55733	https	TLSv1.2	364	Client Hello
17	0.545287	215.255.186.158	251.217.119.1...	55734	https	TLSv1.2	364	Client Hello
22	0.764012	215.255.186.158	251.217.119.1...	55369	https	TLSv1	364	Client Hello
23	0.850374	251.217.119.170	215.255.186.1...	https	55734	TLSv1.2	1514	Server Hello
26	0.858399	251.217.119.170	215.255.186.1...	https	55734	TLSv1.2	645	Certificate, Server Key E
31	0.851933	215.255.186.158	251.217.119.1...	55734	https	TLSv1.2	192	Client Key Exchange, Chan
33	0.908004	215.255.186.158	251.217.119.1...	55681	https	TLSv1	364	Client Hello
34	0.952042	215.255.186.158	251.217.119.1...	55723	https	TLSv1	364	Client Hello
36	1.016351	251.217.119.170	215.255.186.1...	https	55734	TLSv1.2	292	New Session Ticket, Chang
42	1.222375	251.217.119.170	215.255.186.1...	https	55733	TLSv1.2	1514	Server Hello
45	1.222428	251.217.119.170	215.255.186.1...	https	55733	TLSv1.2	645	Certificate, Server Key E
50	1.223959	215.255.186.158	251.217.119.1...	55733	https	TLSv1.2	192	Client Key Exchange, Chan
53	1.387800	251.217.119.170	215.255.186.1...	https	55733	TLSv1.2	292	New Session Ticket, Chang
59	1.534411	215.255.186.158	251.217.119.1...	55736	https	TLSv1.2	364	Client Hello
62	1.541089	215.255.186.158	251.217.119.1...	55737	https	TLSv1	364	Client Hello

> Frame 1: 364 bytes on wire (2912 bits), 364 bytes captured (2912 bits)
> Ethernet II, Src: 06:94:a7:35:af:33 (06:94:a7:35:af:33), Dst: 06:7e:64:b6:bc:3f (06:7e:64:b6:bc:3f)
> Internet Protocol Version 4, Src: 215.255.186.158, Dst: 251.217.119.170
> Transmission Control Protocol, Src Port: 55726 (55726), Dst Port: https (443), Seq: 1215393884, Ack: 3100448357,
> Transport Layer Security

- When it comes to SSL/TLS handshakes, you should remember two things:
 - Each SSL/TLS handshake is effectively a **new connection** (consuming resources)
 - SSL/TLS handshakes are quite **CPU intensive operations** (server-side)
- The number of new Client Hello messages is **abnormal**.
- it looks like we are dealing with a **TLS Renegotiation Attack (DoS attack against the TLS layer)**

SMTP (Simple Mail Transfer Protocol)

How SMTP works ?

- It's the protocol responsible for sending emails
- SMTP is a text-based protocol, meaning that it relies on **exchanging ASCII based strings as commands** between the **server** and the **client**.

HELO	Sent by a client to identify itself, usually with a domain name.
MAIL FROM	Identifies the sender of the message; used in the form MAIL FROM::
RCPT TO	Identifies the message recipients; used in the form RCPT TO::
VRFY	Verifies that a mailbox is available for message delivery::

- The SMTP server starts the conversation, once the **TCP three-way handshake is completed**, by sending its banner, containing the server's name and version.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000			TCP	62	1077->25 [SYN] Seq=0 Win=16384 Len=0 MS
2	0.000000			TCP	62	25->1077 [SYN, ACK] Seq=0 Ack=1 Win=175
3	0.020029			TCP	60	1077->25 [ACK] Seq=1 Ack=1 Win=17520 Len=0
4	0.020029			SMTP	158	S: 220 [REDACTED]
5	0.030043			SMTP	67	C: EHLO Client
6	0.190274			TCP	54	25->1077 [ACK] Seq=105 Ack=14 Win=17507
7	0.420605			SMTP	290	S: 250 Server Hello [REDACTED] 250
8	0.430619			SMTP	66	C: AUTH LOGIN
9	0.430619			SMTP	72	S: 334 VXNlcmbWU6
10	0.430619			SMTP	64	C: User: [REDACTED]
11	0.430619			SMTP	72	S: 334 UGFzc3dvcmQ6
12	0.430619			SMTP	64	C: Pass: [REDACTED]
13	0.440634			SMTP	91	S: 235 2.7.0 Authentication successful

Malicious SMTP Traffic Example

- The lack of proper security configuration may also allow the attacker to connect to the SMTP server and **manually enumerate** the users on that server using the **VRFY**, **EXPN** or **RCPT TO** commands.
- User enumeration may be used as part of a **social engineering attack** or as a first step of a brute force attack against account passwords on that server.

DNS (Domain Name System)

- resolves **names** to **IP addresses**.
- DNS is a **query-response** protocol.
- DNS traffic normally uses **UDP** on port **53**.
- DNS traffic should go to **DNS servers only**.

Normal vs Malicious DNS Traffic

Normal DNS Traffic	Suspicious DNS Traffic
Port 53, UDP	Traffic on port 53 but using TCP instead of UDP.
Should only go to DNS Servers.	DNS traffic not going to DNS Servers.
Should see DNS Responses to DNS Queries.	A lot of DNS Queries with no DNS responses or vice versa.

Normal DNS Traffic

dns					
No.	Time	Source	Destination	Protocol	Length Info
16	26.200151138	172.16.5.100	172.16.5.10	DNS	83 Standard query 0xde40 PTR 5.5.16.172.in-addr.arpa
19	26.272980431	172.16.5.10	172.16.5.100	DNS	127 Standard query response 0xde40 PTR 5.5.16.172.in-addr.arpa PTR wkst-techsupport.sportsfoo.com
41	56.605405613	172.16.5.100	172.16.5.10	DNS	94 Standard query 0xa620 PTR 5.5.16.172.in-addr.arpa OPT
42	56.639661726	172.16.5.10	172.16.5.100	DNS	138 Standard query response 0xa620 PTR 5.5.16.172.in-addr.arpa PTR wkst-techsupport.sportsfoo.com OPT

- 4 packets: 2 packets for DNS Queries and 2 for DNS Responses.

▶ Frame 16: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0
▶ Ethernet II, Src: b2:fe:ed:db:02:32 (b2:fe:ed:db:02:32), Dst: VMware_a1:a4:5f (00:50:56:a1:a4:5f)
▶ Internet Protocol Version 4, Src: 172.16.5.100, Dst: 172.16.5.10
▶ User Datagram Protocol, Src Port: 42653 [b]Dst Port: 53
▼ Domain Name System (query)
[Response In: 19]
Transaction ID: 0xde40
Flags: 0x0100 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
▼ Queries
▼ 5.5.16.172.in-addr.arpa: type PTR, class IN
Name: 5.5.16.172.in-addr.arpa
[Name Length: 23]
0000 00 50 56 a1 a4 5f b2 fe ed db 02 32 08 00 45 00 .PV.2..E.
0010 00 45 ae 11 00 00 40 11 6a 08 ac 10 05 64 ac 10 .E....@. j....d..
0020 05 0a a6 9d 00 35 00 31 2d 00 de 40 01 00 00 015.1 ...@....
0030 00 00 00 00 00 01 35 01 35 02 31 36 03 31 375 .5.16.17
0040 32 07 69 6e 2d 61 64 64 72 04 61 72 70 61 00 00 2.in-add r.arpa..
0050 0c 00 01

- this is a **UDP** packet and it's using an expected port, **53**.

```

▶ Frame 19: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits) on interface 0
▶ Ethernet II, Src: VMware_a1:a4:5f (00:50:56:a1:a4:5f), Dst: b2:fe:ed:db:02:32 (b2:fe:ed:db:02:32)
▶ Internet Protocol Version 4, Src: 172.16.5.10, Dst: 172.16.5.100
▶ User Datagram Protocol, Src Port: 53, Dst Port: 42653
▼ Domain Name System (response)
  [Request In: 16]
  [Time: 0.072829293 seconds]
  Transaction ID: 0xde40
  ▶ Flags: 0x8580 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  ▼ Queries
    ▼ 5.5.16.172.in-addr.arpa: type PTR, class IN
      Name: 5.5.16.172.in-addr.arpa
      [Name Length: 23]
      [Label Count: 6]
      Type: PTR (domain name PoinTeR) (12)
      Class: IN (0x0001)
  ▼ Answers
    ▼ 5.5.16.172.in-addr.arpa: type PTR, class IN, wkst-techsupport.sportsfoo.com
      Name: 5.5.16.172.in-addr.arpa
      Type: PTR (domain name PoinTeR) (12)
      Class: IN (0x0001)
      Time to live: 3600
      Data length: 32
      Domain Name: wkst-techsupport.sportsfoo.com

```

- This is the DNS Response to the DNS Query in packet looks normal

Malicious DNS Traffic

- **DNS Zone Transfers** : a way to **replicate DNS databases** across a group of **DNS servers**.
- **DNS tunnels**