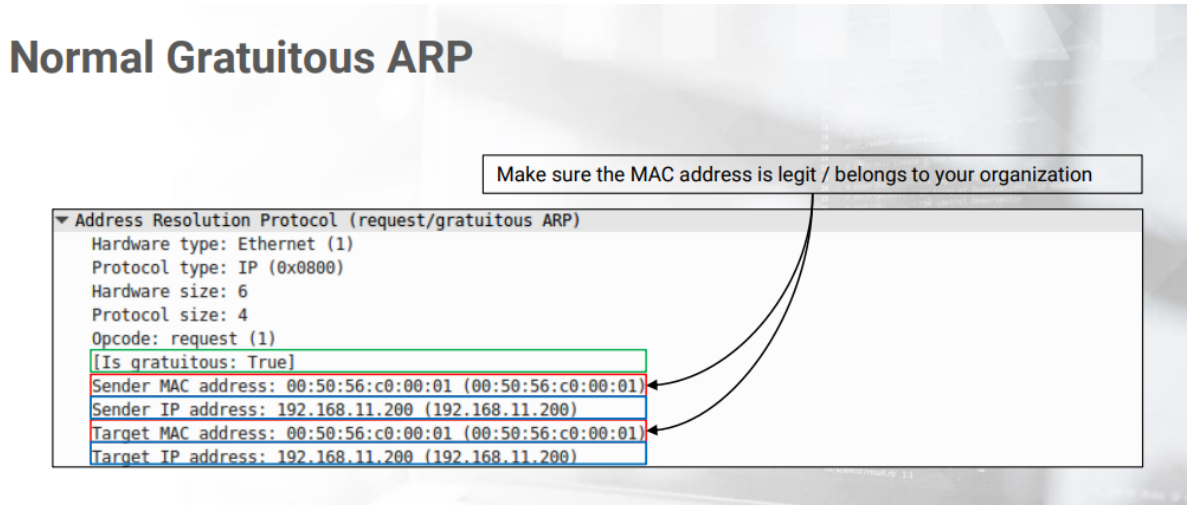
2. Network Traffic & Flow Analysis

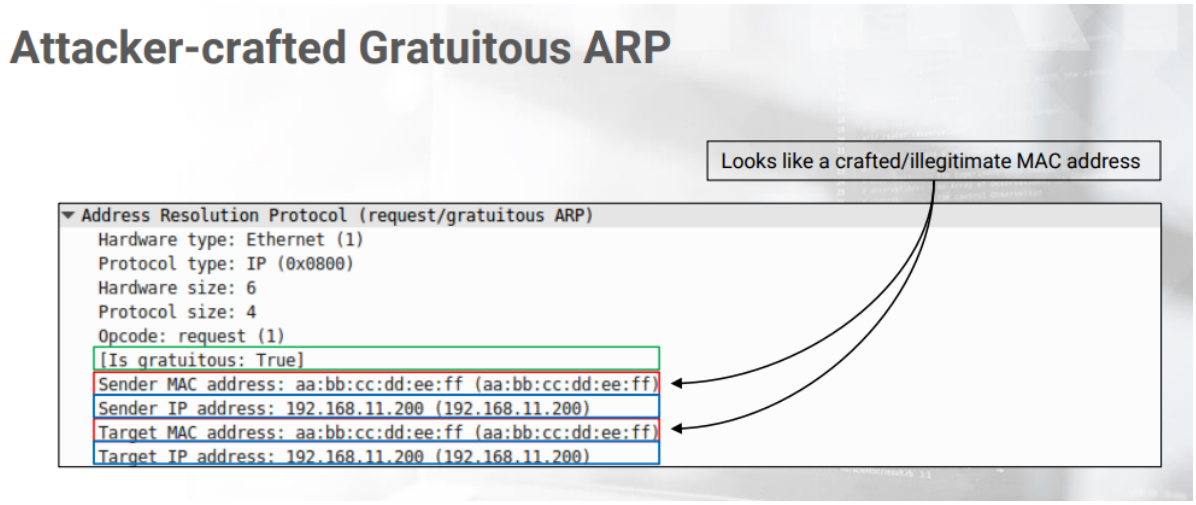
Data Link Layer

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.**

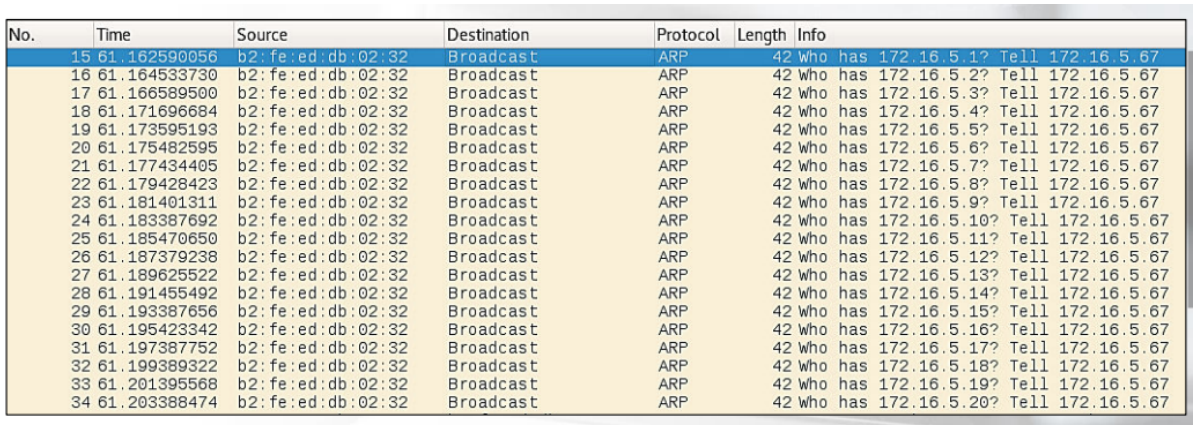




- **Gratuitous ARP reply**: It is an ARP reply that has been sent **without being requested**. (usually malicious)

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



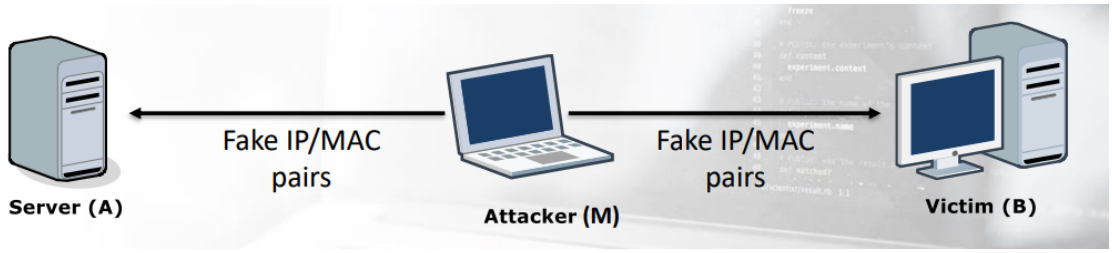
No.	Time	Source	Destination	Protocol	Length	Info
15	61.162590056	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.1? Tell 172.16.5.67
16	61.164533730	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.2? Tell 172.16.5.67
17	61.166589500	b2:fe:ed:db:02:32	Broadcast	ARP	42	Who has 172.16.5.3? Tell 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)

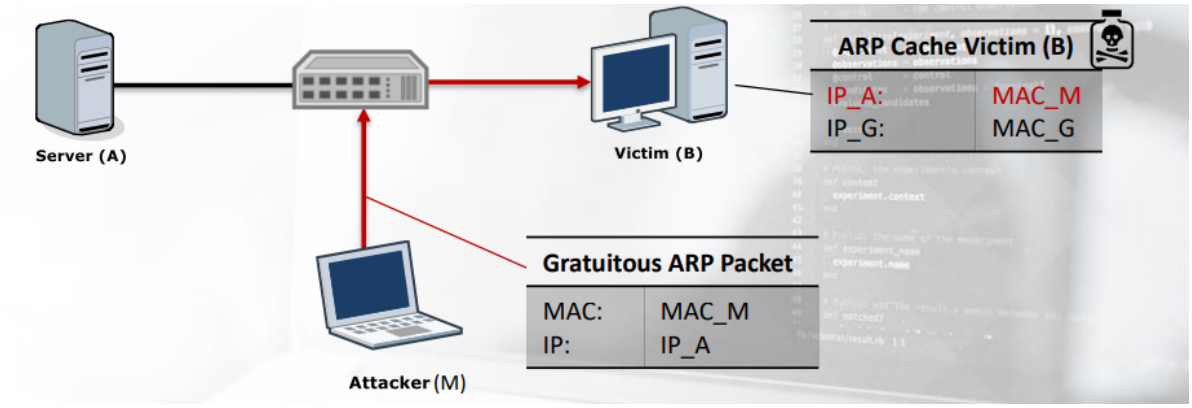
- ARP poisoning can be exploited to add fake information between
- The following are the steps for a successful attack (**ARP Poisoning):**

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**



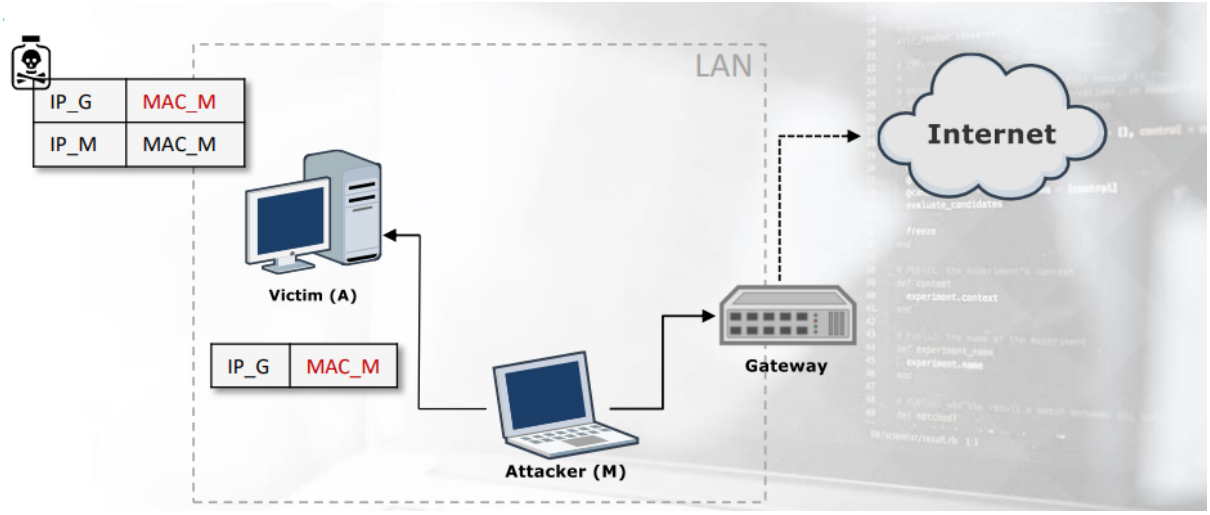
- Because of the **TTL** in hosts **ARP caches**, an attacker would need to send these packets at intervals lower than the timeout (usually every **30** seconds is a good choice).
- Once the gratuitous ARP packet is sent, B's ARP cache gets poisoned with the entry: `IP_A->MAC_M`. Next time B wants to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

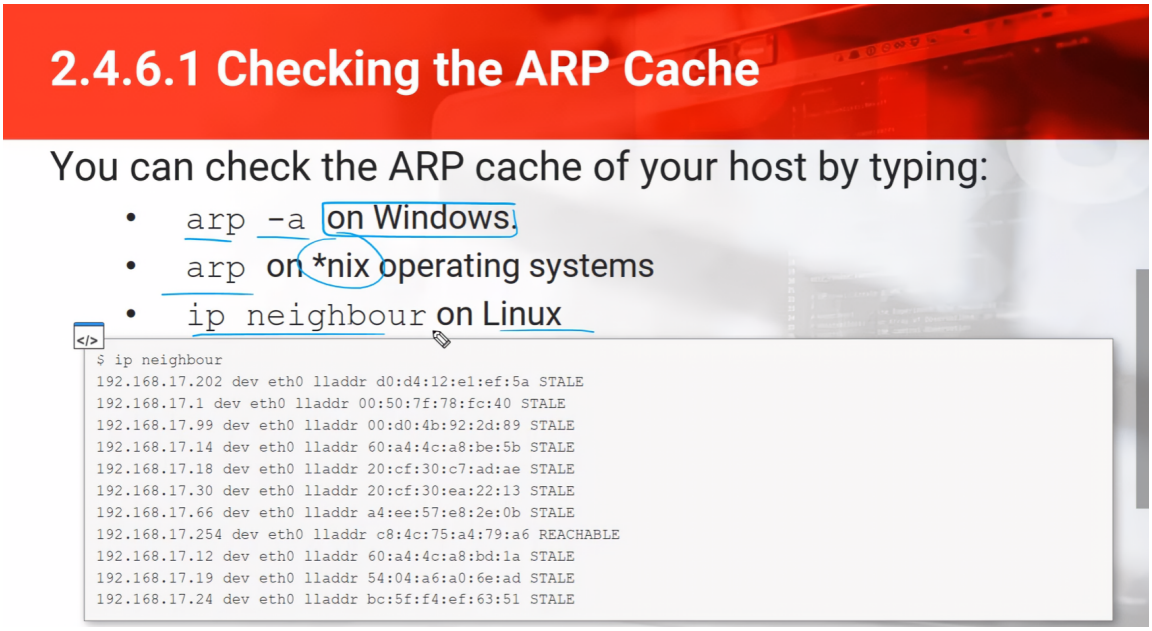


The following describes the steps that take place in the previous scenario:

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)



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 to 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.

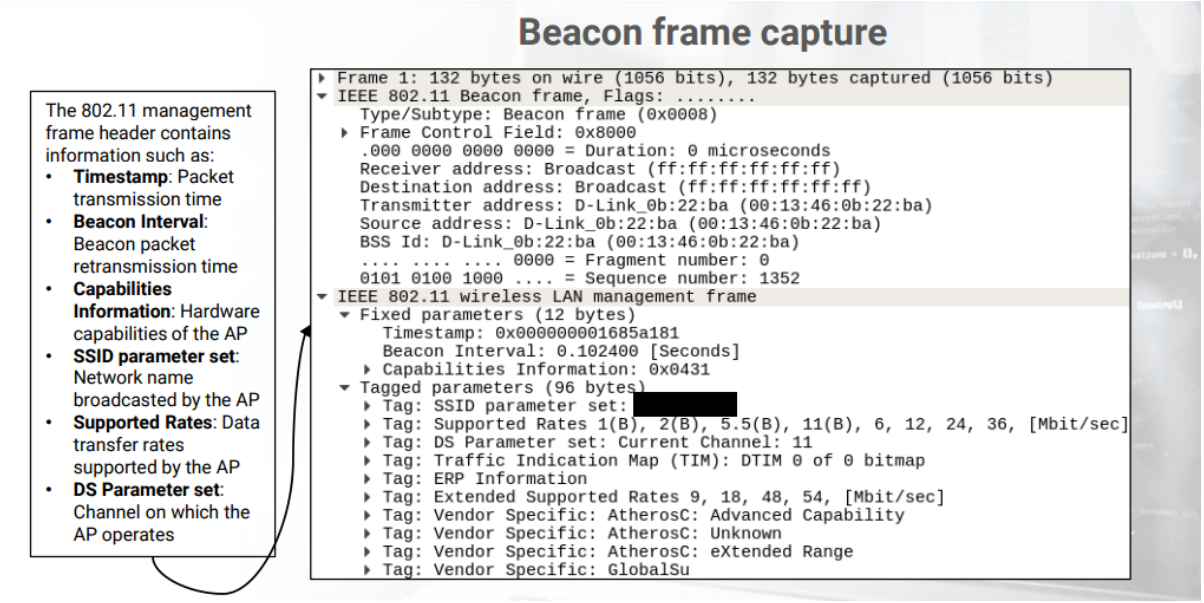
802.11 Wireless (layer 2) header

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.

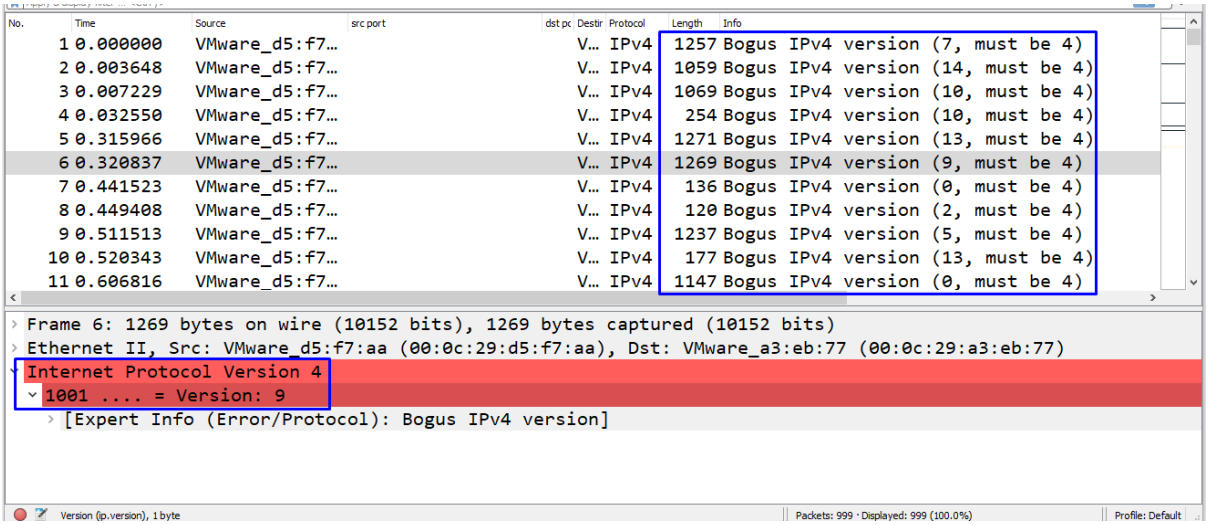
- **Beacon packets** are broadcasted from a wireless access point to inform other listening wireless clients of its existence and its connection requirements.



IP Layer Attacks with each header

IDS/Firewall Evasion (invalid IP version)

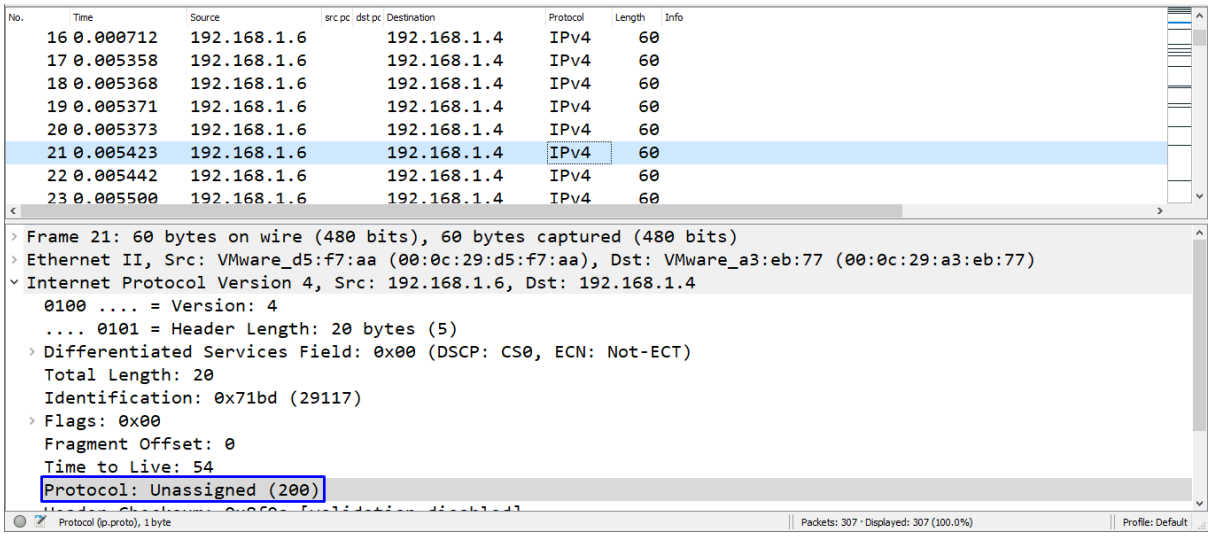
- Oftentimes, attackers check the reactions of firewalls and IDS by crafting and sending datagrams with an **invalid IP version**

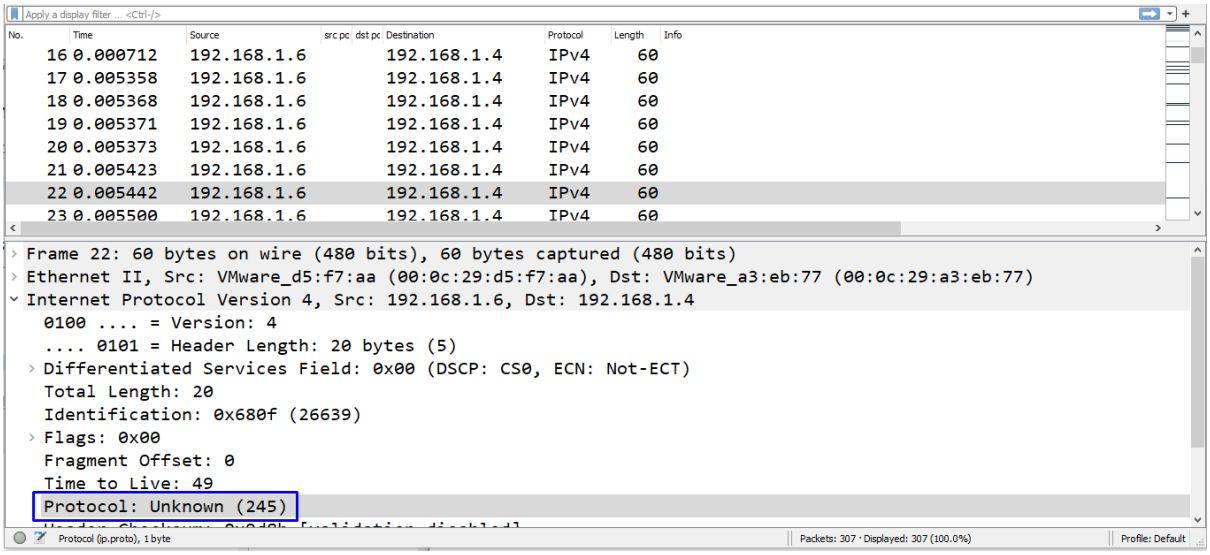


Stealthy Nmap Scan (changing the IPv4 Protocol Number.)

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.





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)

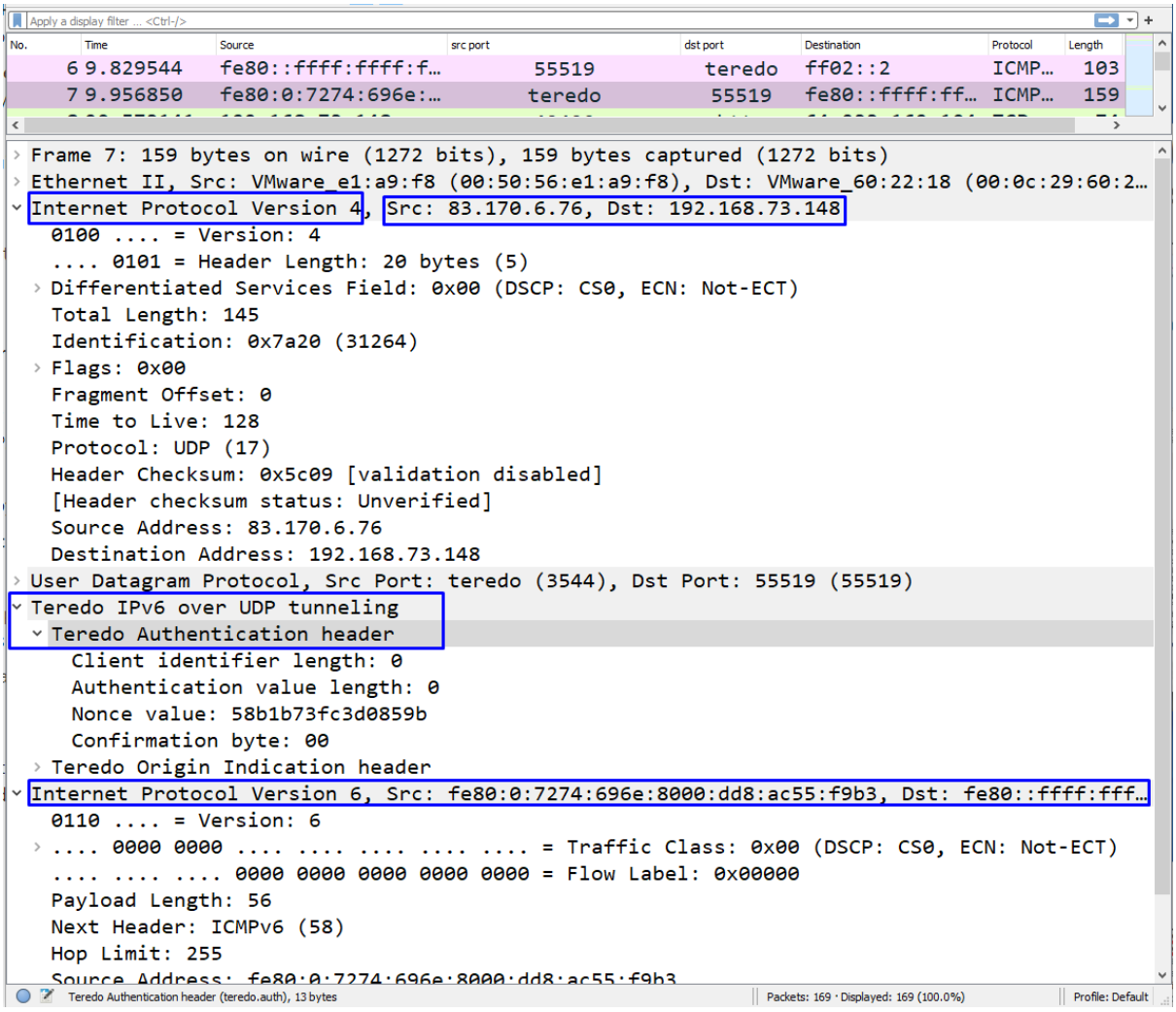
- Fragmentation is the action of **dividing a packet** whose size is greater than the Maximum Transmission Unit (MTU)
- it can be performed by a **router** or the sending **host** itself.
- Each fragment's IP header contains fields and values that facilitate **reassembling** the original packet at the 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.



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

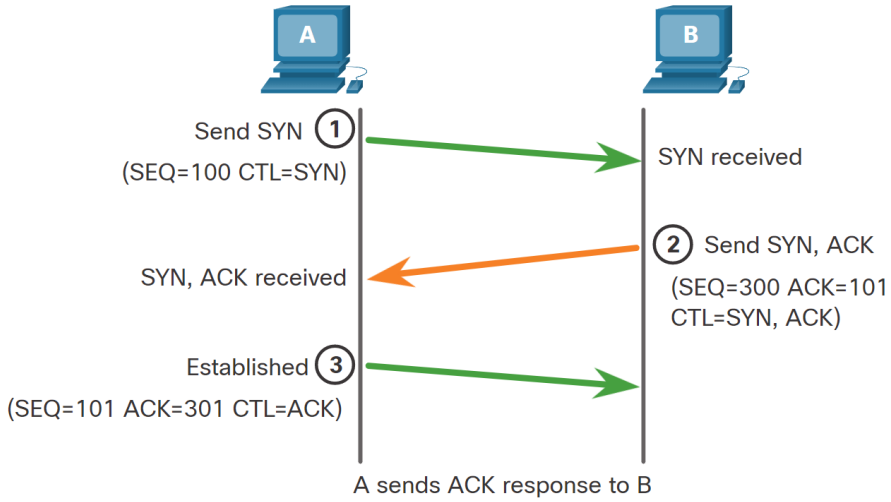
TCP Server Processes

- 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.

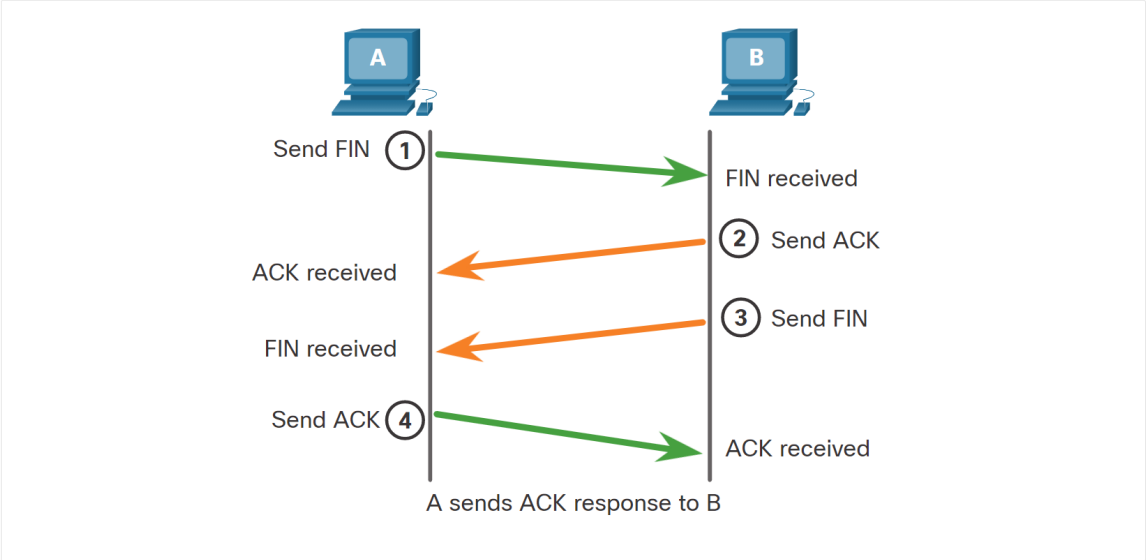
- There can be many ports open simultaneously on a server, one for each active server application.

TCP Connection Establishment (3-way Handshake)

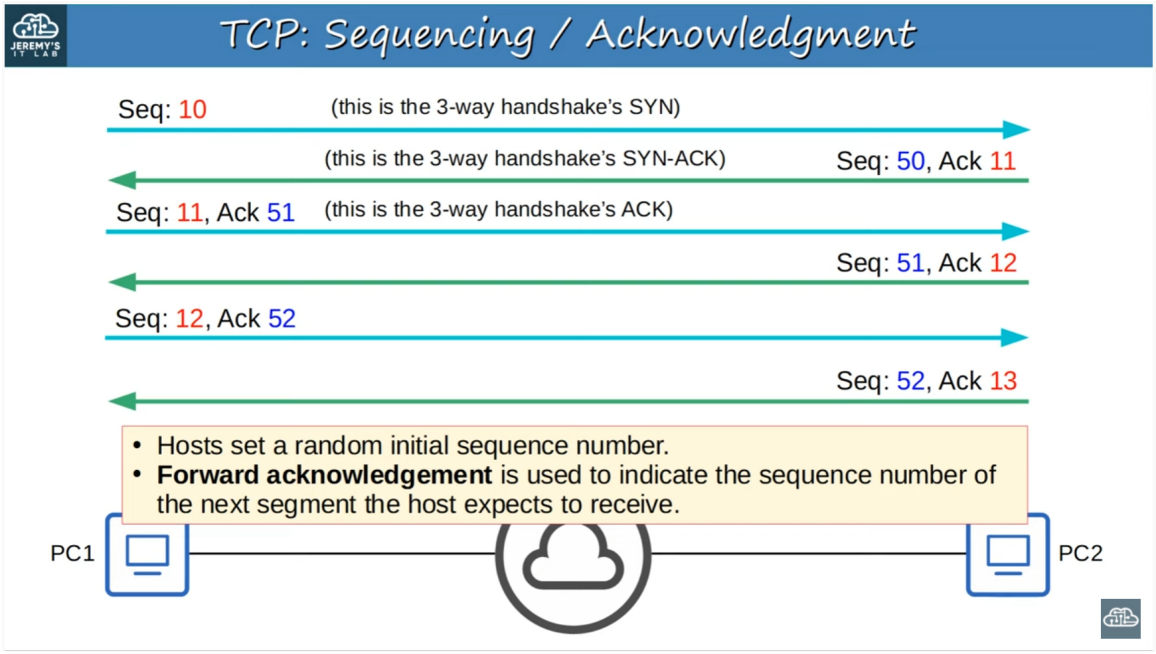
- It establishes that the destination device is present on the network.
- It verifies that the destination device has an active service and is accepting requests on the destination port number that the initiating client intends to use.
- It informs the destination device that the source client intends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



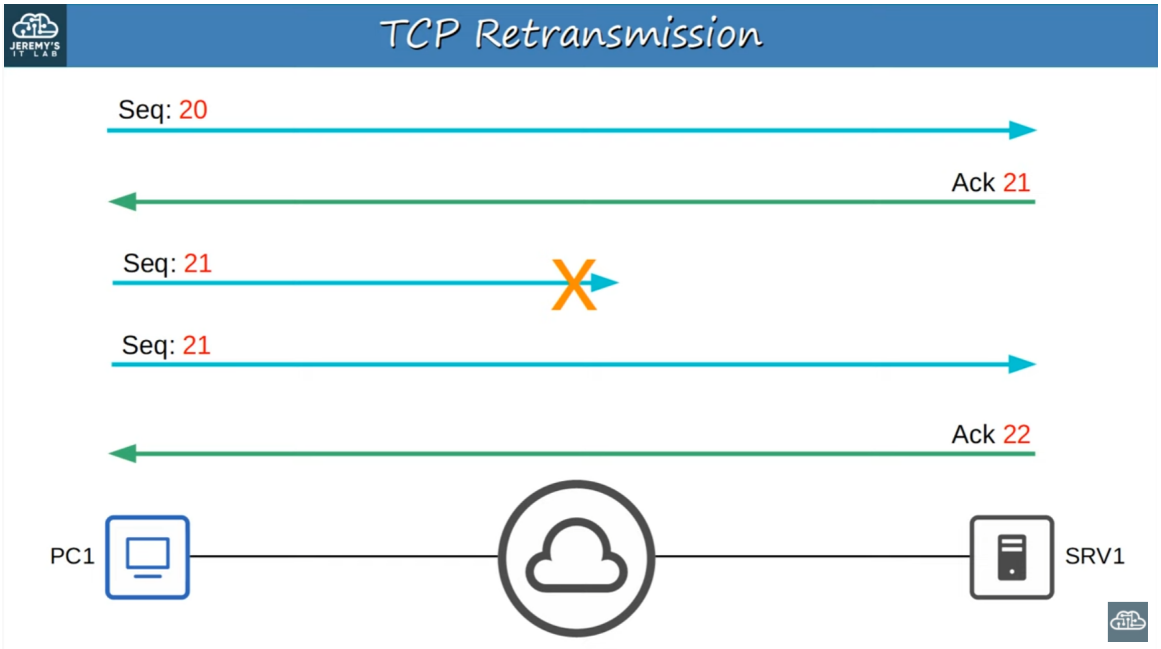
Session Termination (4-way Handshake)



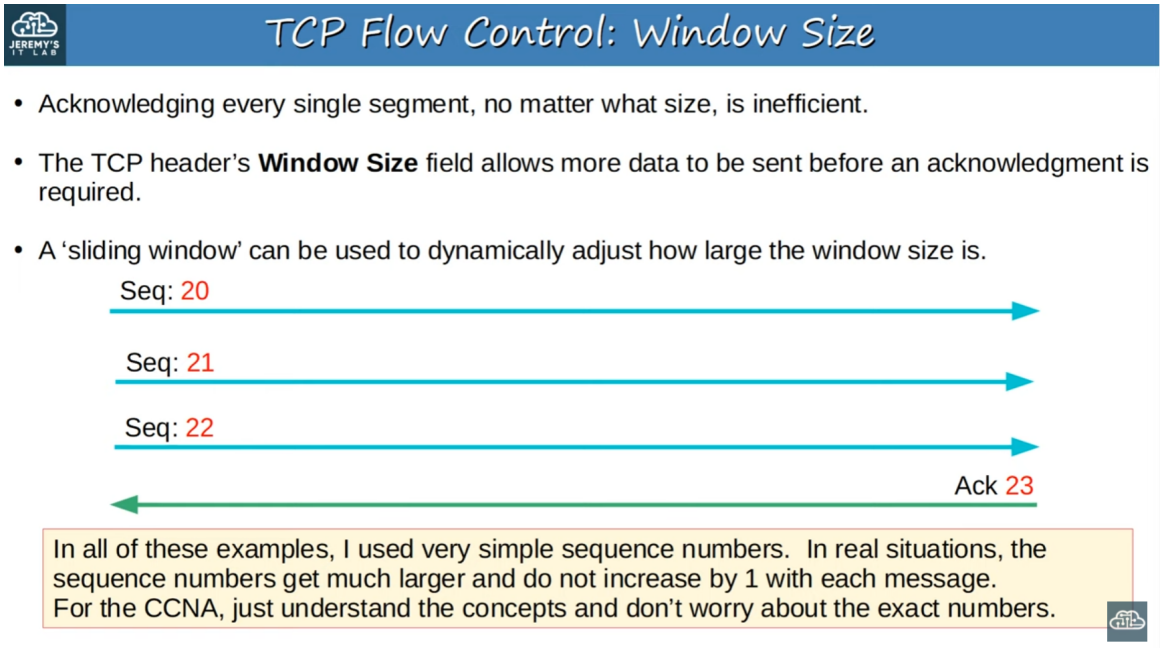
TCP Reliability – Sequence Numbers and Acknowledgements



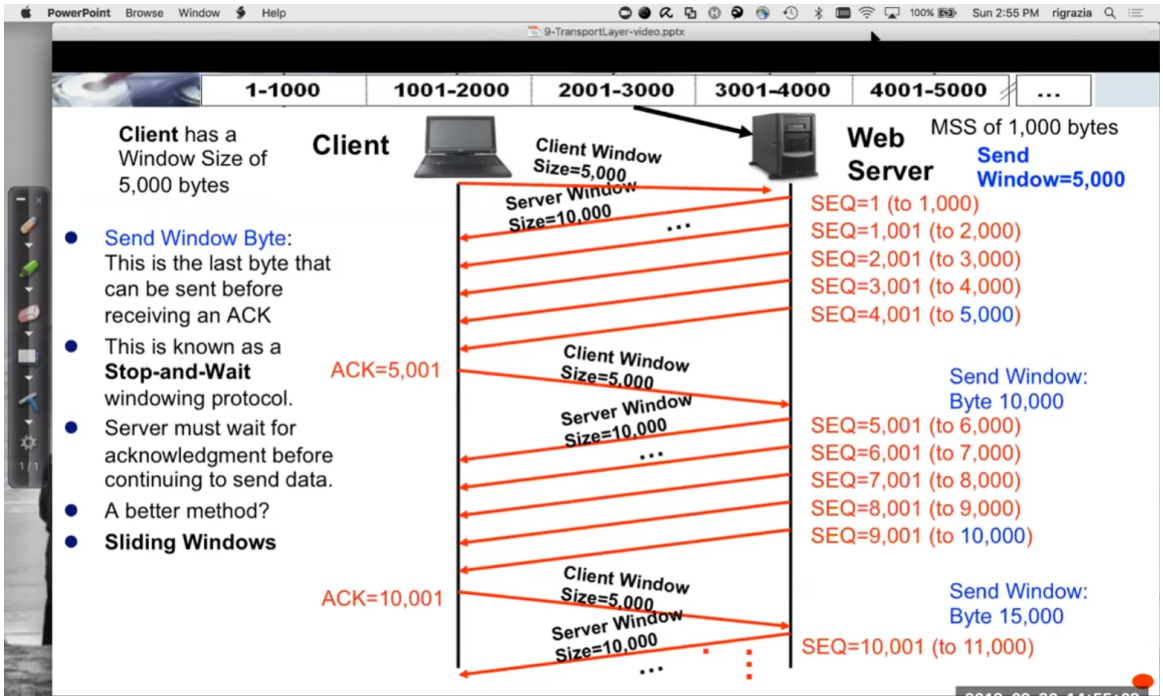
TCP Retransmission



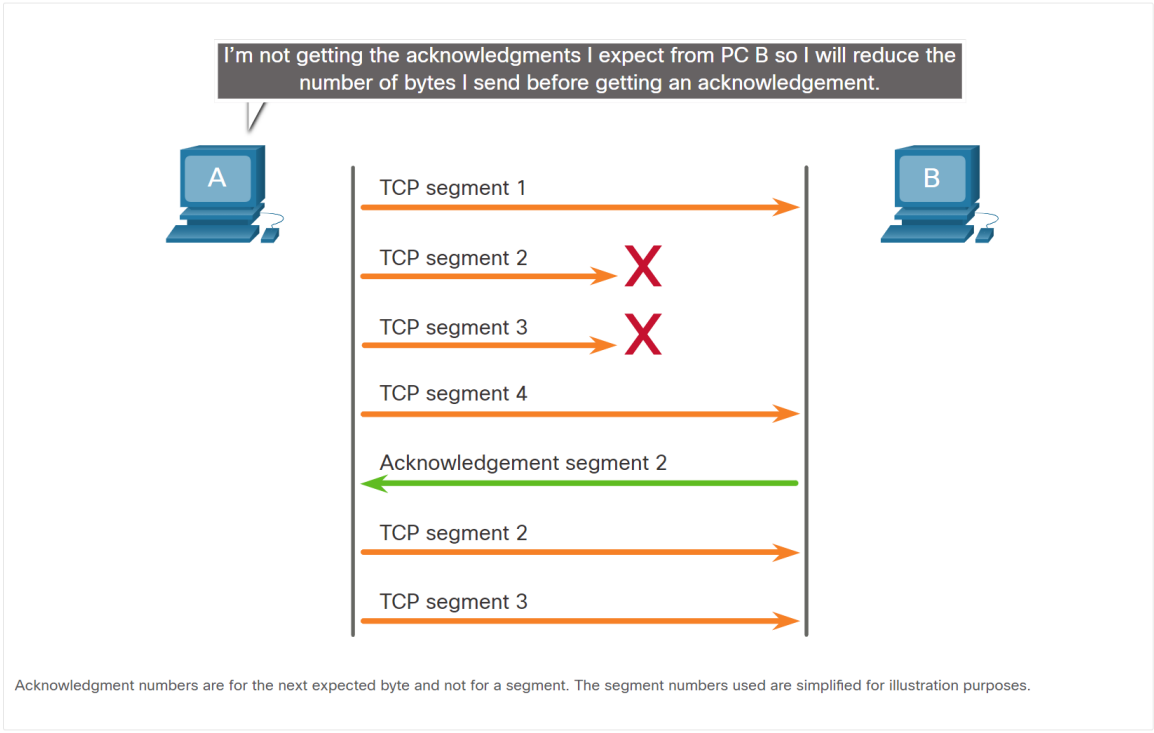
TCP Flow Control - Window Size (Watch)



MSS= Maximum Segment Size



TCP Congestion Control



Acknowledgment numbers are for the next expected byte and not for a segment. The segment numbers used are simplified for illustration purposes.

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.

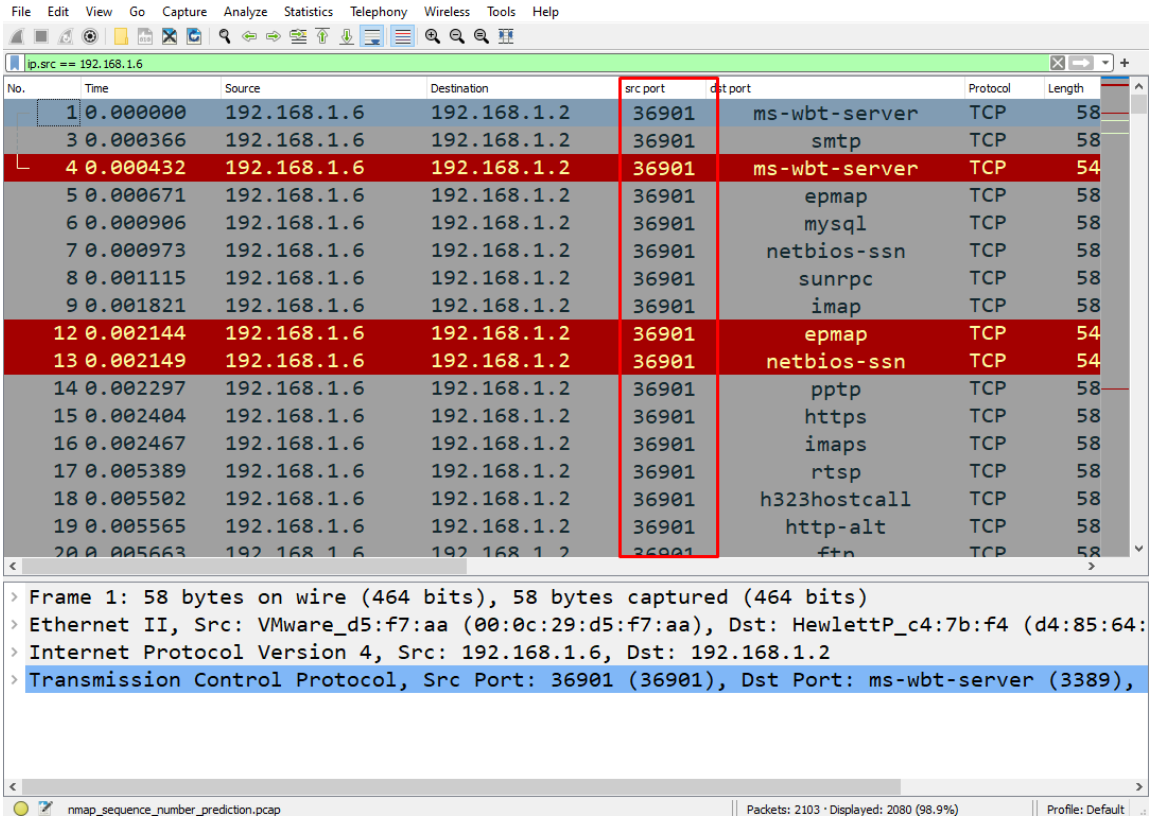
UDP Header Field	Description
Source Port	A 16-bit field used to identify the source application by port number.
Destination Port	A 16-bit field used to identify the destination application by port number.
Length	A 16-bit field that indicates the length of the UDP datagram header.
Checksum	A 16-bit field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

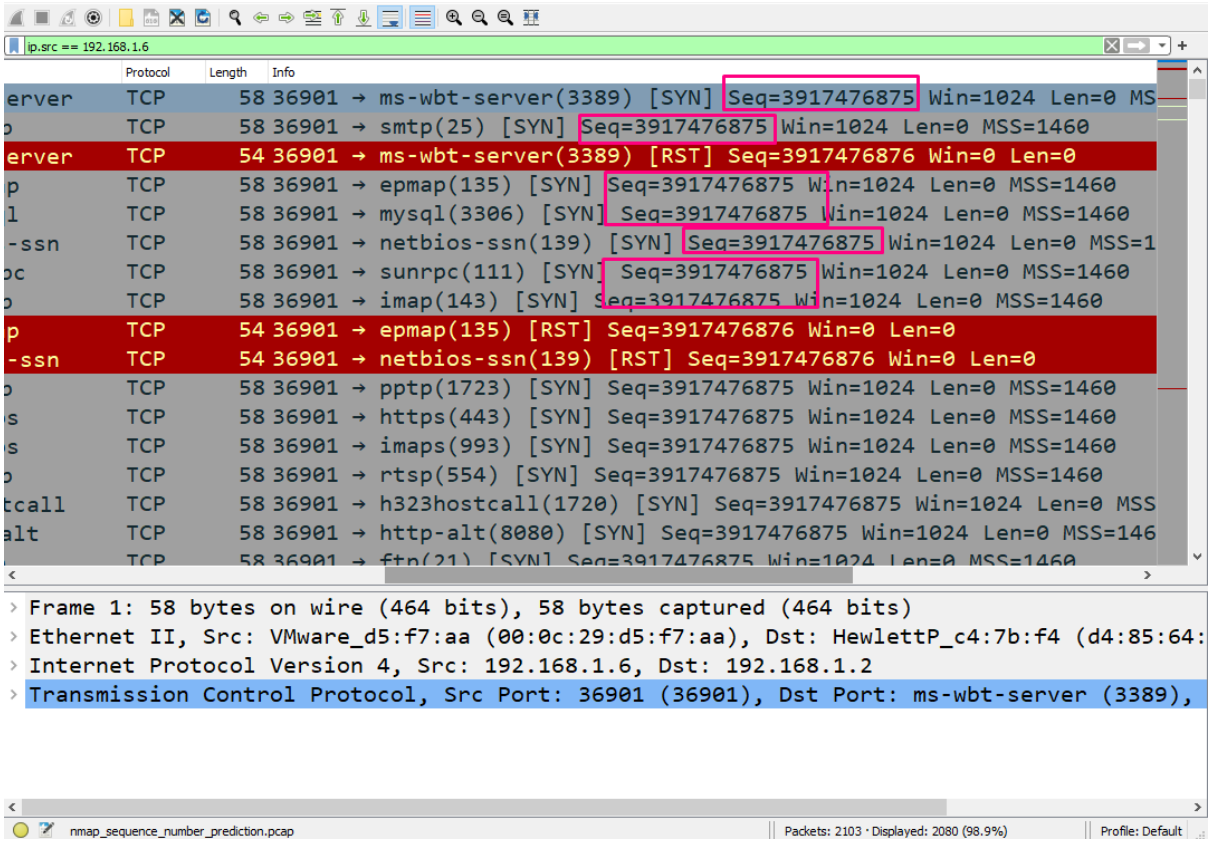
1. **Excessive SYN** packets (scanning)
2. Usage of different **flags**
3. **Single host to multiple ports** or single host to **multiple nodes** (scanning)
4. **Source Port Abnormalities :**
    - **Privileged** (**server**) ports [**1-1023**] ← Should remain unchanged during the entire connection
    - **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.

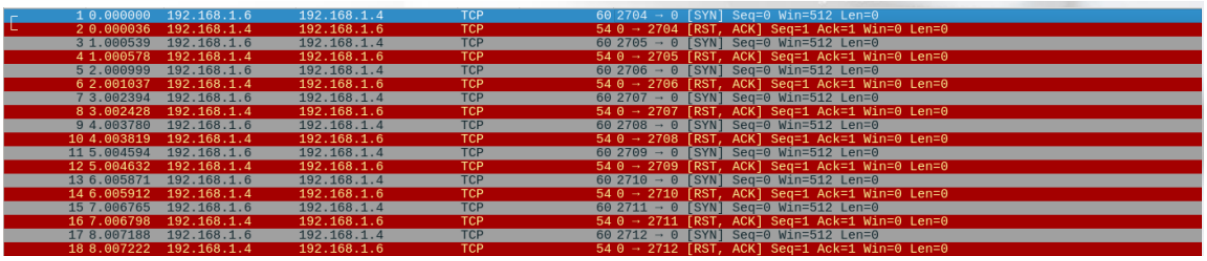
5. **Sequence Number Prediction & SYN Scanning**

- 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**

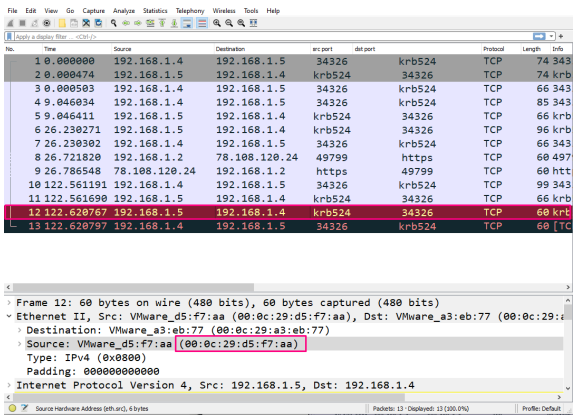
- when sending a **SYN request** to **Port 0** if the host responds with **RST, ACK** then it's **alive**
    - if not it will not respond

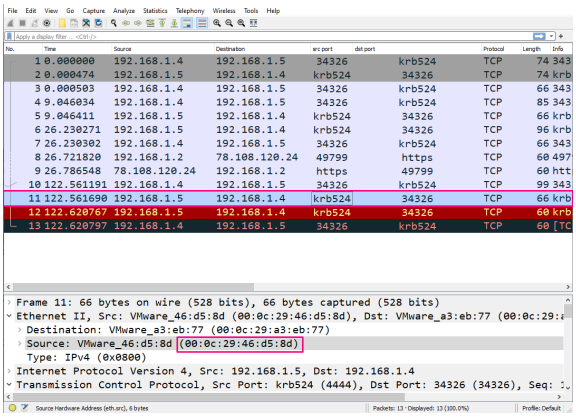


6. **TCP RST Attack**

- 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**

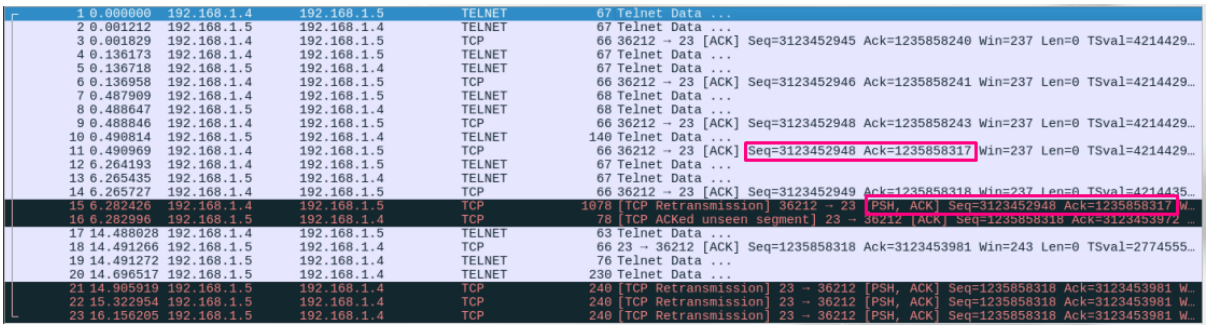




- **the attacker spoofed the src ip and sent a TCP RST packet to the server ti cut the connection.**

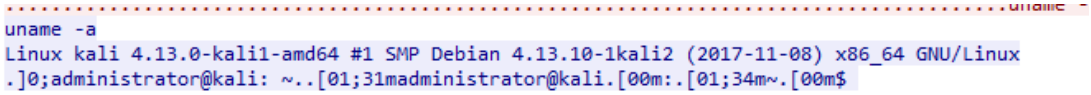
7. **TCP Session Hijacking**

- 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.)



- 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.



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 disscused)

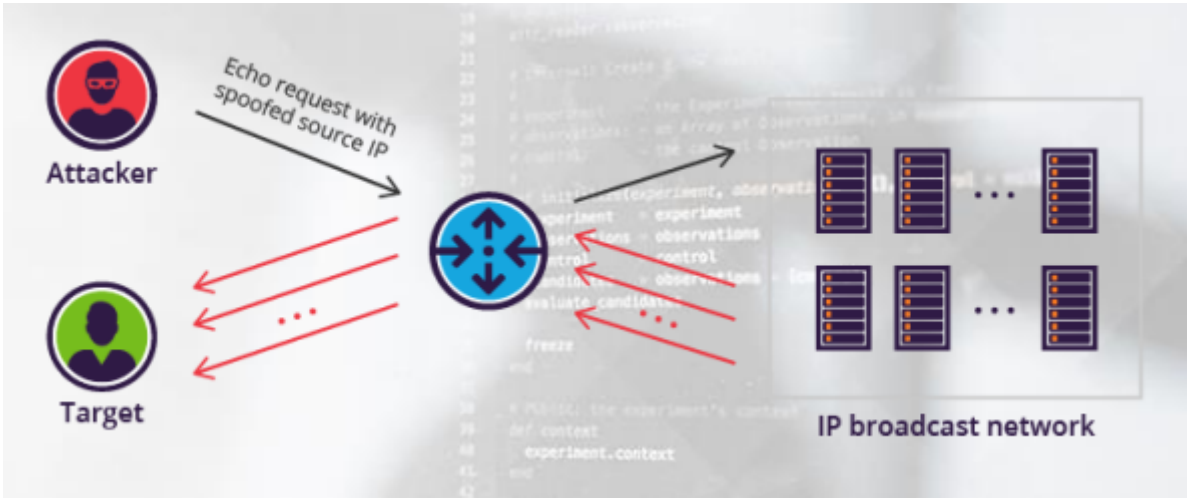
ICMP Abuse

1. **ICMP Echo Request (8)**

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

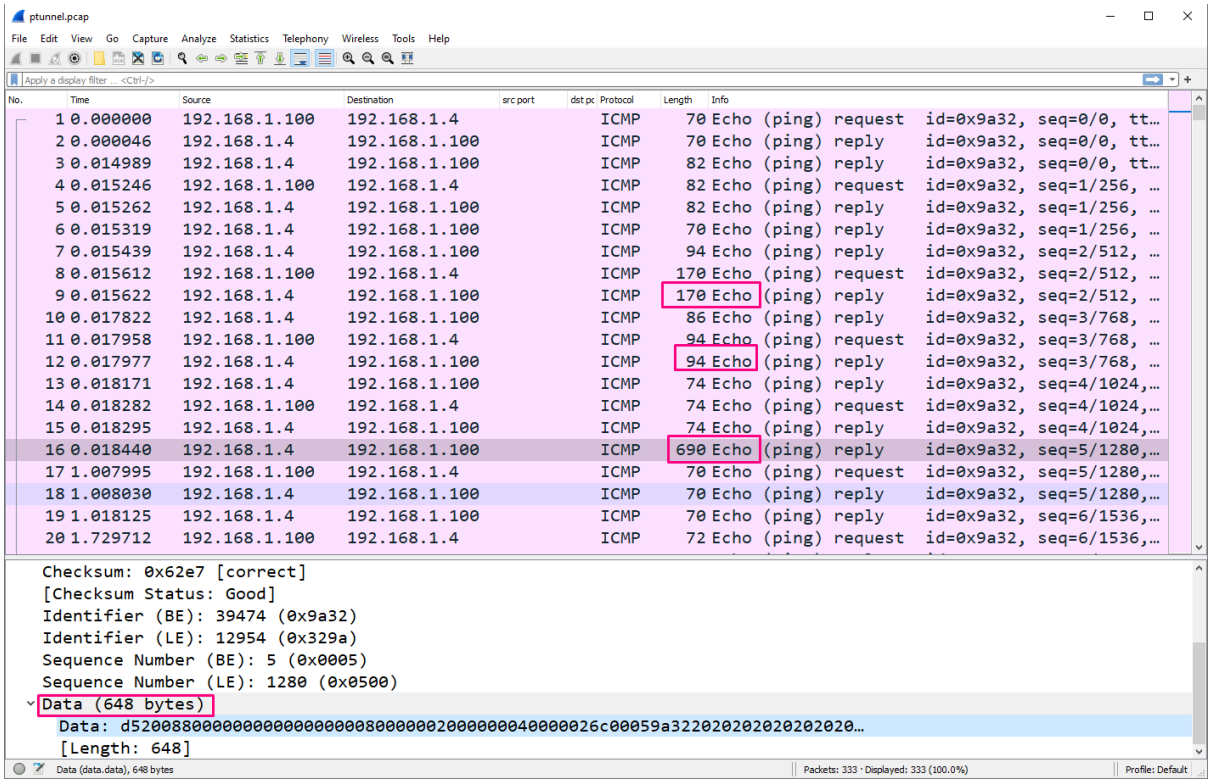
2. **ICMP Address Mask Request(17)/Reply(18)**
   - can be used to identify a target host's subnet mask.
3. **ICMP Timestamp Request(13)/Reply (14)**
   - can be used to obtain **timestamps** from remote systems.
   - used in zero days attack to see if the system got patched
4. **Smurf Attack(DDoS)**

- 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



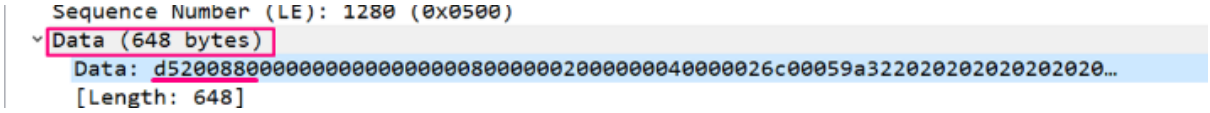
5. **ICMP Tunneling**

- 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.



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.



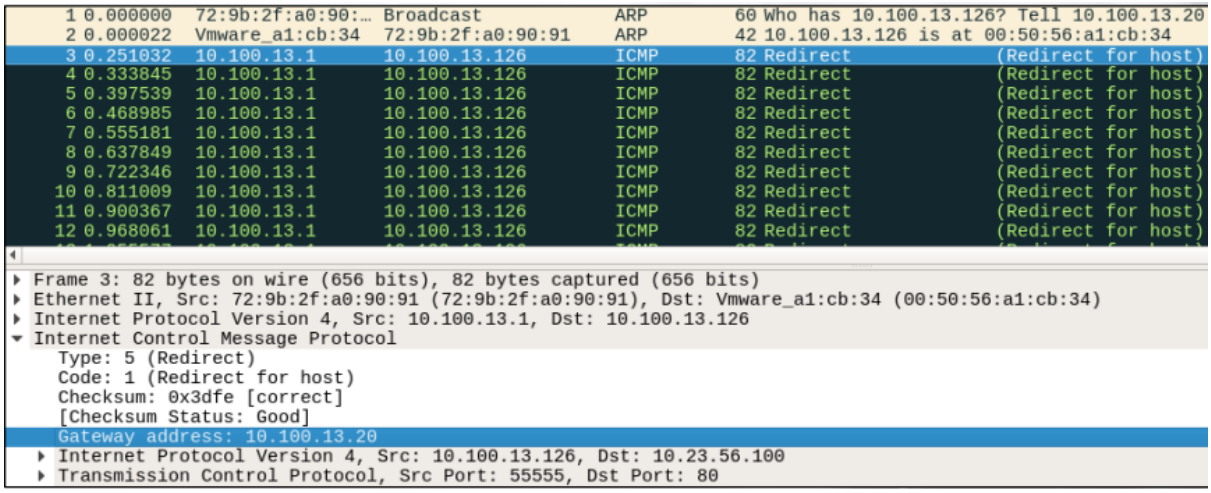
- we can use `strings ptunnel.pcap` to identified the strings in the pcap

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.



- 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 :

1. **(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.

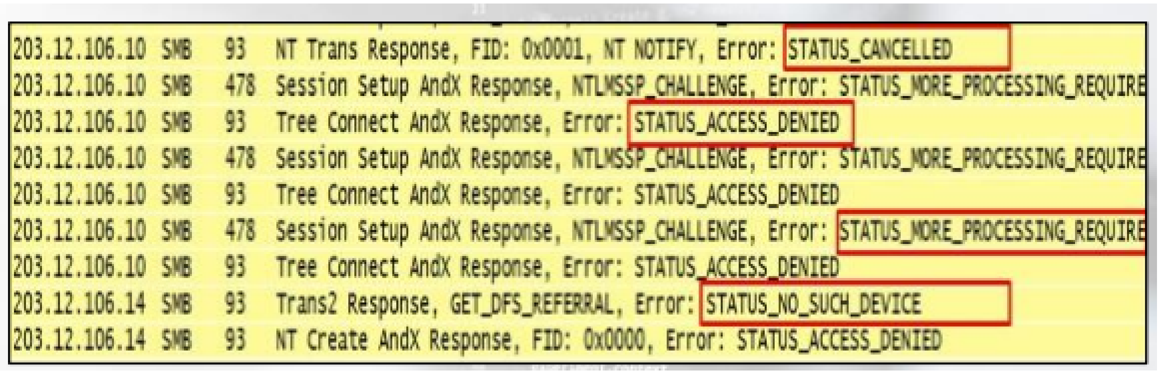
- There are additional protocols such as Server Message Block (**SMB**)

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.

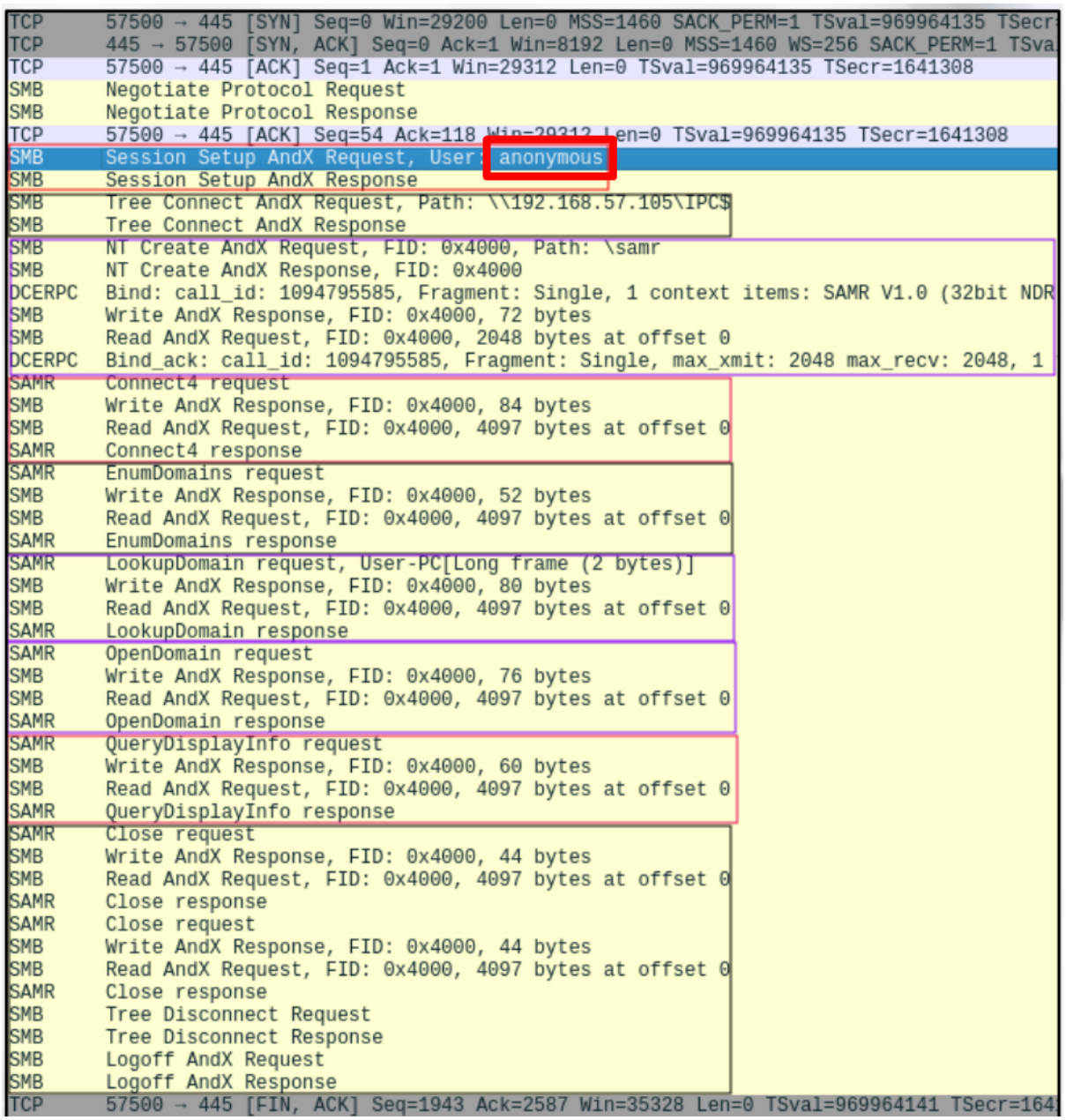
Detection

- **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_REQUIRE
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_REQUIRE
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_REQUIRE
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.

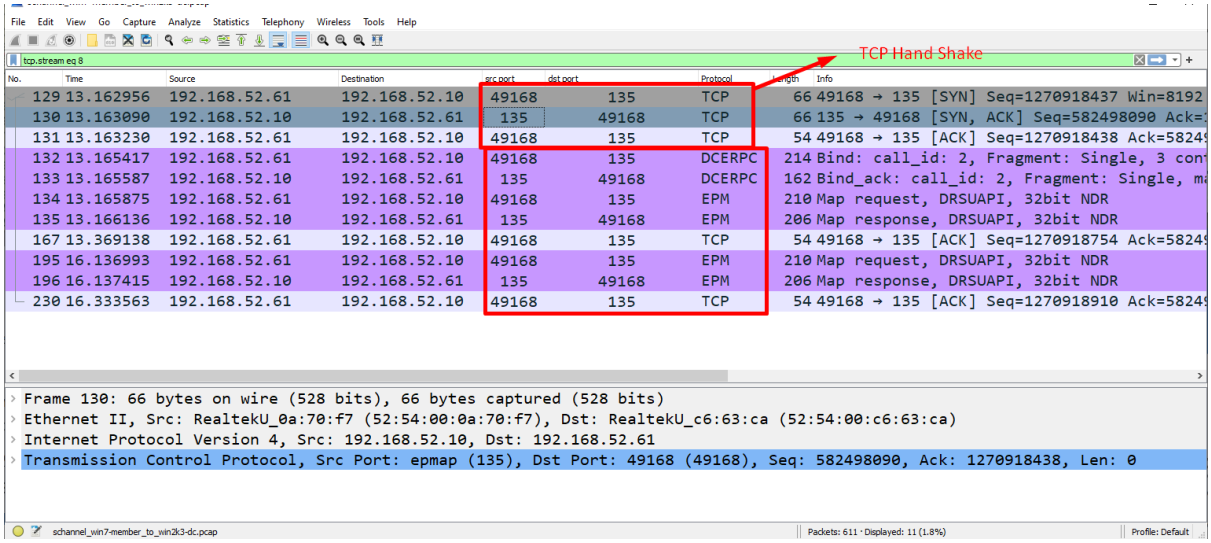


MSRPC

- 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:

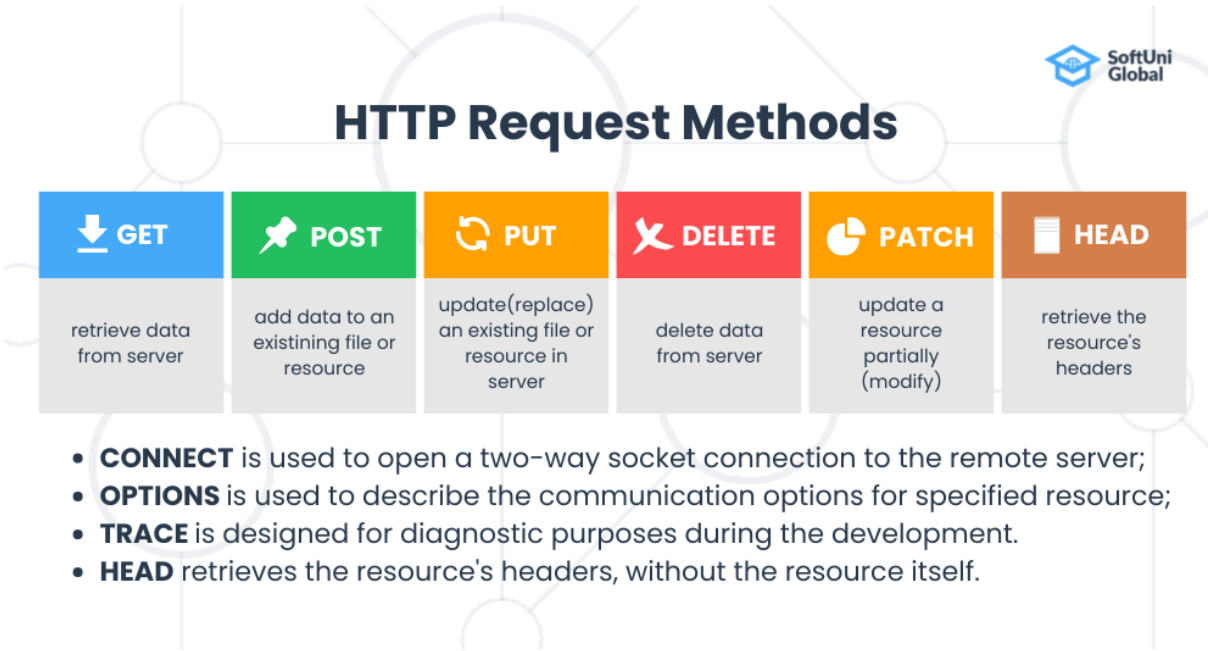
- RPC over **SMB**
- DCOM (RPC **directly over TCP/UDP**) [TCP/UDP port 135]



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

HTTP

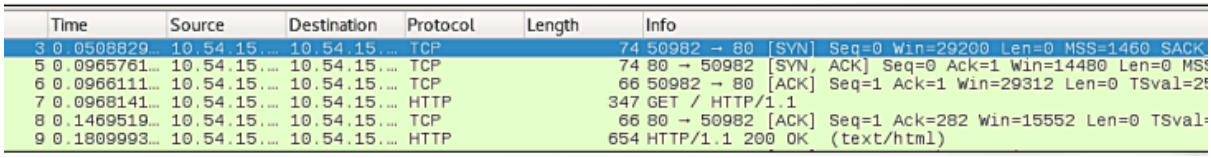
- HTTP uses methods to perform various operations.
  - Not all methods will be permitted by web server.



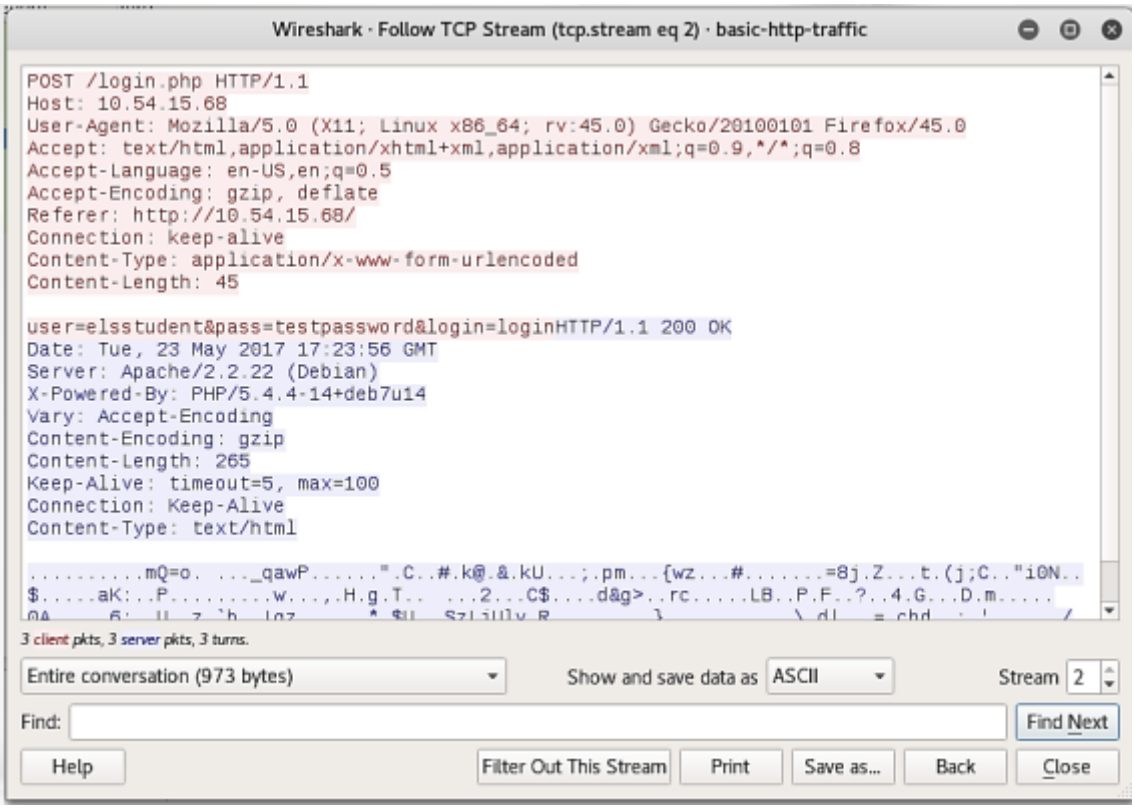
Normal vs Suspicious HTTP Trafic

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, web shells, etc. will use this port because typically in all corporate environments the port is open.
Plaintext traffic	If the traffic is encrypted 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 IP address instead of FQDN format.

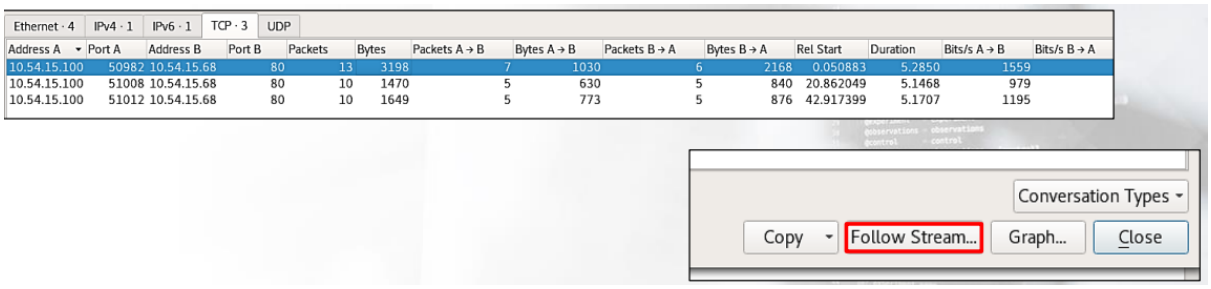
Normal HTTP Traffic example:



- 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**



- 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.

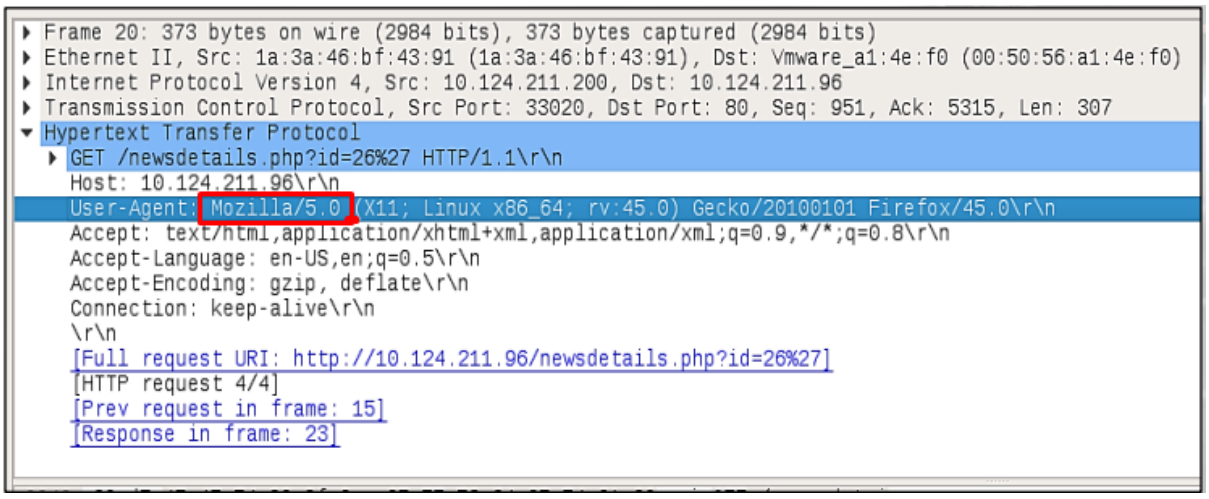


Malicious HTTP Traffic Example

- 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	66	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	66	33020 → 80 [ACK] Seq=1258 Ack=6642 Win=44800 Len=0 TSval=127992 TSecr=224575
10.124.211.96	10.124.211.200	TCP	66	80 → 33020 [FIN, ACK] Seq=6642 Ack=1258 Win=18768 Len=0 TSval=225826 TSecr=127992
10.124.211.200	10.124.211.96	TCP	66	33020 → 80 [FIN, ACK] Seq=1258 Ack=6643 Win=44800 Len=0 TSval=129243 TSecr=225826
10.124.211.96	10.124.211.200	TCP	66	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=225
10.124.211.200	10.124.211.96	TCP	66	33022 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=129989 TSecr=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	66	80 → 33022 [ACK] Seq=1 Ack=324 Win=15552 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=30336 Len=0 TSval=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=22659
10.124.211.200	10.124.211.96	TCP	66	33022 → 80 [ACK] Seq=324 Ack=1344 Win=33280 Len=0 TSval=130008 TSecr=226591
10.124.211.200	10.124.211.96	TCP	66	33022 → 80 [FIN, ACK] Seq=324 Ack=1344 Win=33280 Len=0 TSval=131258 TSecr=226591
10.124.211.96	10.124.211.200	TCP	66	80 → 33022 [FIN, ACK] Seq=1344 Ack=324 Win=15552 Len=0 TSval=227842 TSecr=130008
10.124.211.200	10.124.211.96	TCP	66	33022 → 80 [ACK] Seq=325 Ack=1345 Win=33280 Len=0 TSval=131259 TSecr=227842
10.124.211.96	10.124.211.200	TCP	66	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=228
10.124.211.200	10.124.211.96	TCP	66	33024 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=131898 TSecr=228481
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	66	80 → 33024 [ACK] Seq=1 Ack=324 Win=15552 Len=0 TSval=228492 TSecr=131899
10.124.211.96	10.124.211.200	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

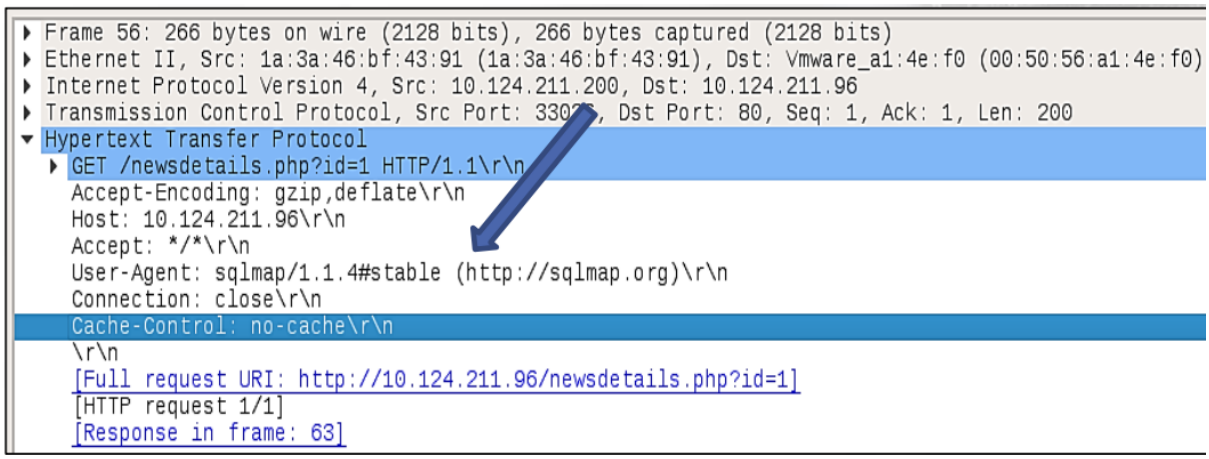
- By further inspection in packet **#20**, we see that the **User-Agent** is **Firefox** and the OS is **Linux**



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

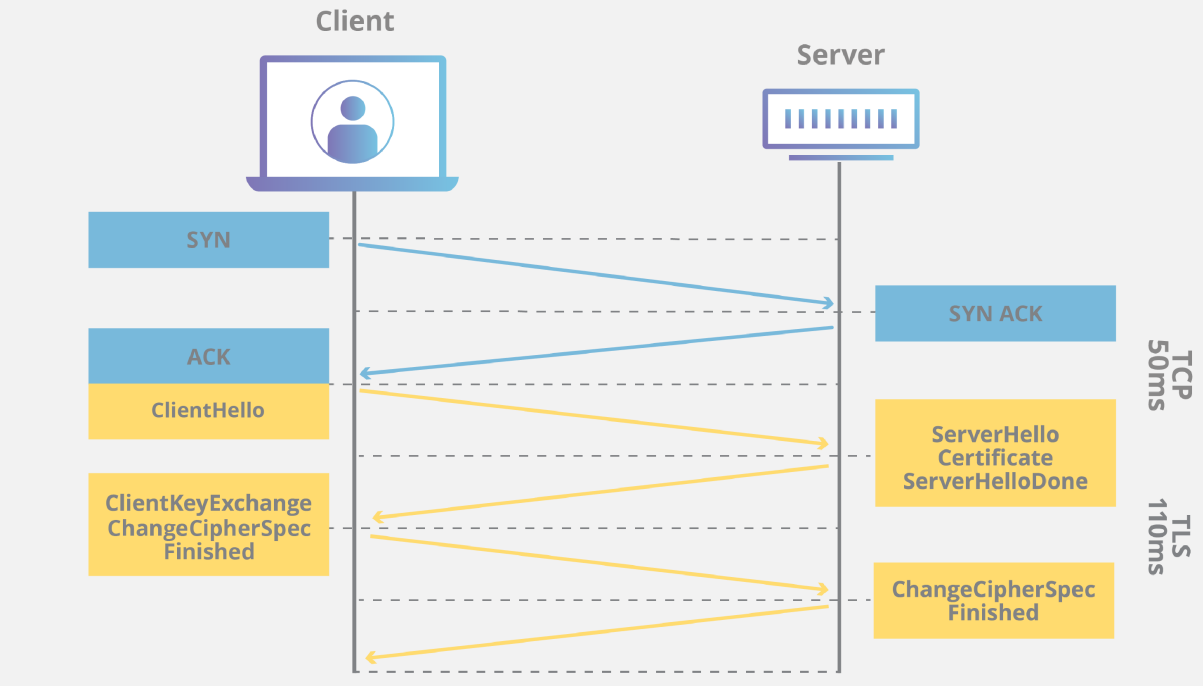
Time	Source	Destination	Protocol	Length	Info
56 34.163959	10.124.211.200	10.124.211.96	HTTP	266	GET /newsdetails.php?id=1 HTTP/1.1
57 34.169322	10.124.211.96	10.124.211.200	TCP	74	80 → 33028 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_PERM=1 TSval=230739 TSecr=134145 WS=4
58 34.169364	10.124.211.200	10.124.211.96	TCP	66	33028 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=134156 TSecr=230739
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	66	80 → 33026 [ACK] Seq=1 Ack=201 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	10.124.211.200	10.124.211.96	TCP	66	33026 → 80 [ACK] Seq=201 Ack=1326 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	10.124.211.200	10.124.211.96	TCP	66	33026 → 80 [ACK] Seq=201 Ack=1342 Win=32128 Len=0 TSval=134165 TSecr=230748
65 34.206508	10.124.211.96	10.124.211.200	TCP	66	80 → 33026 [FIN, ACK] Seq=1342 Ack=201 Win=15552 Len=0 TSval=230748 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=230748
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	66	80 → 33026 [ACK] Seq=1343 Ack=202 Win=15552 Len=0 TSval=230758 TSecr=134165
71 34.269905	10.124.211.96	10.124.211.200	TCP	74	80 → 33032 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_PERM=1 TSval=230764 TSecr=134171 WS=4
72 34.269934	10.124.211.200	10.124.211.96	TCP	66	33032 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=134181 TSecr=230764
73 35.126264	10.124.211.200	10.124.211.96	HTTP	266	GET /newsdetails.php?id=1 HTTP/1.1
74 35.126569	10.124.211.200	10.124.211.96	TCP	74	33034 → 80 [SYN] Seq=0 Win=29200 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 35.168835	10.124.211.200	10.124.211.96	TCP	66	33028 → 80 [ACK] Seq=201 Ack=1326 Win=32128 Len=0 TSval=134405 TSecr=230988
80 35.168926	10.124.211.96	10.124.211.200	HTTP	82	HTTP/1.1 200 OK (text/html)
81 35.168943	10.124.211.200	10.124.211.96	TCP	66	33028 → 80 [ACK] Seq=201 Ack=1342 Win=32128 Len=0 TSval=134405 TSecr=230988
82 35.168962	10.124.211.96	10.124.211.200	TCP	66	80 → 33028 [FIN, ACK] Seq=1342 Ack=201 Win=15552 Len=0 TSval=230988 TSecr=134395
83 35.170154	10.124.211.200	10.124.211.96	TCP	66	33028 → 80 [FIN, ACK] Seq=201 Ack=1343 Win=32128 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**.



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.

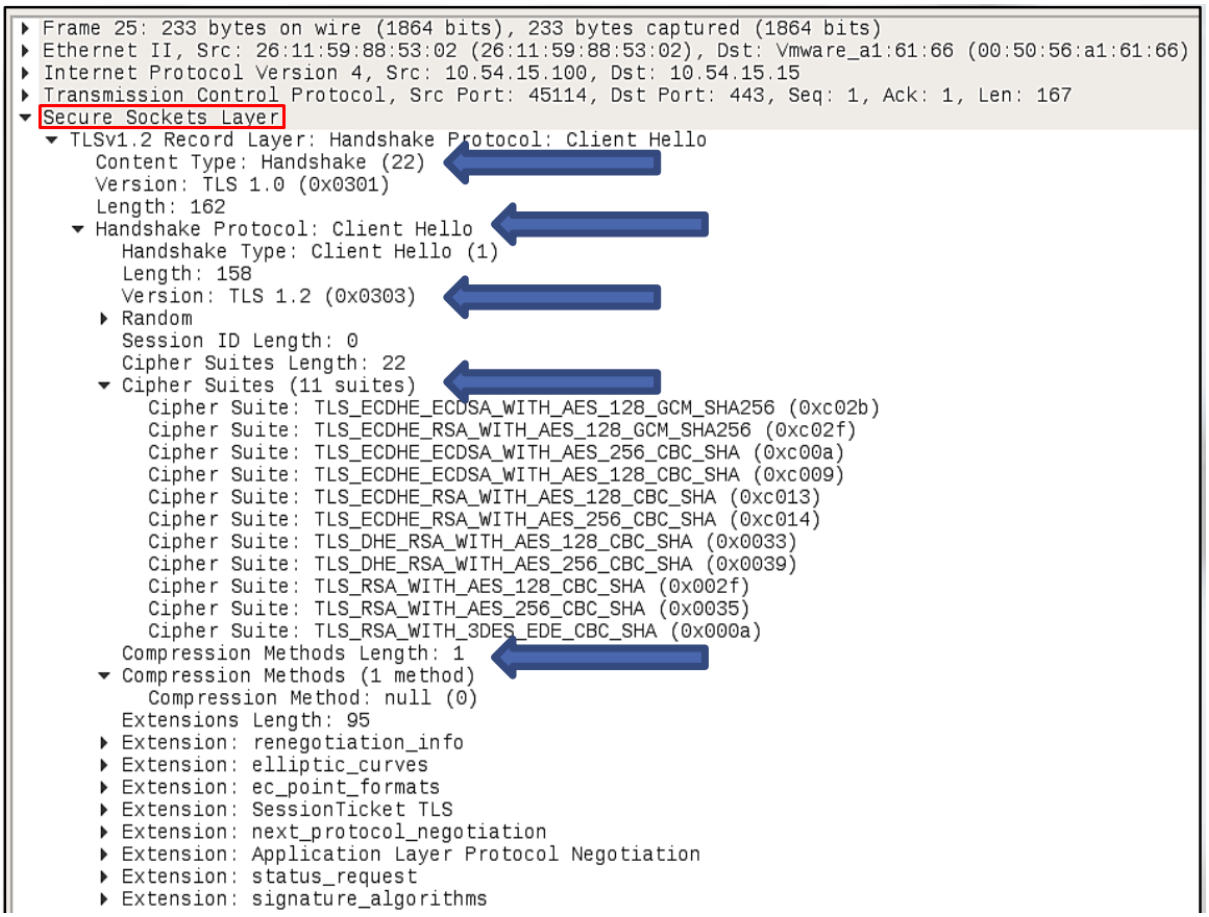


Normal vs Suspicious HTTPS Trafic

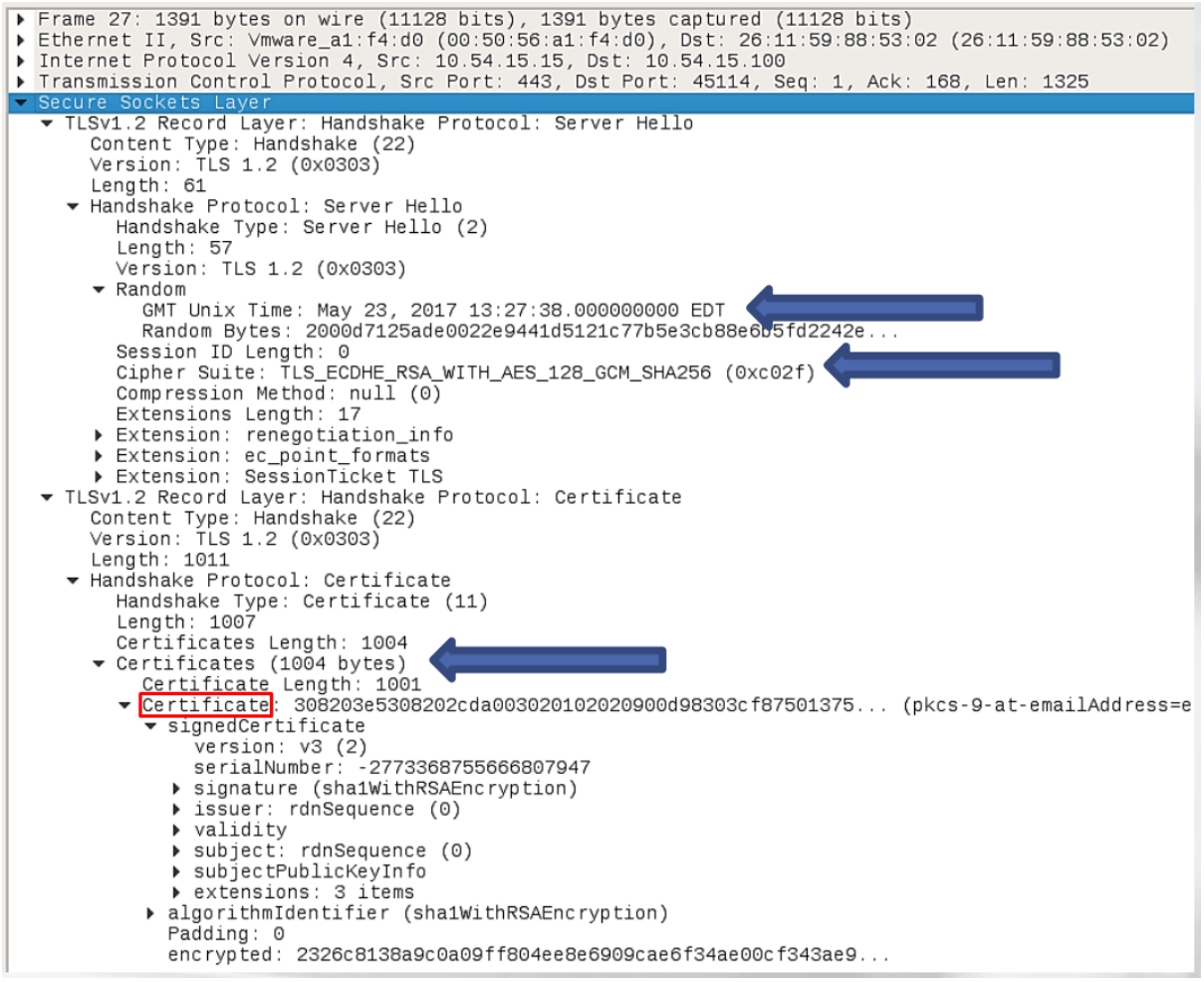
Normal HTTPS Traffic	Suspicious HTTPS Traffic
Port 443, TCP Port 8443, TCP (used as alternate)	Malicious binaries (backdoors), scripts, web shells, etc. will use this port because typically in all corporate environments the port is open.
Encrypted traffic	If the traffic is not encrypted and Secure Sockets 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

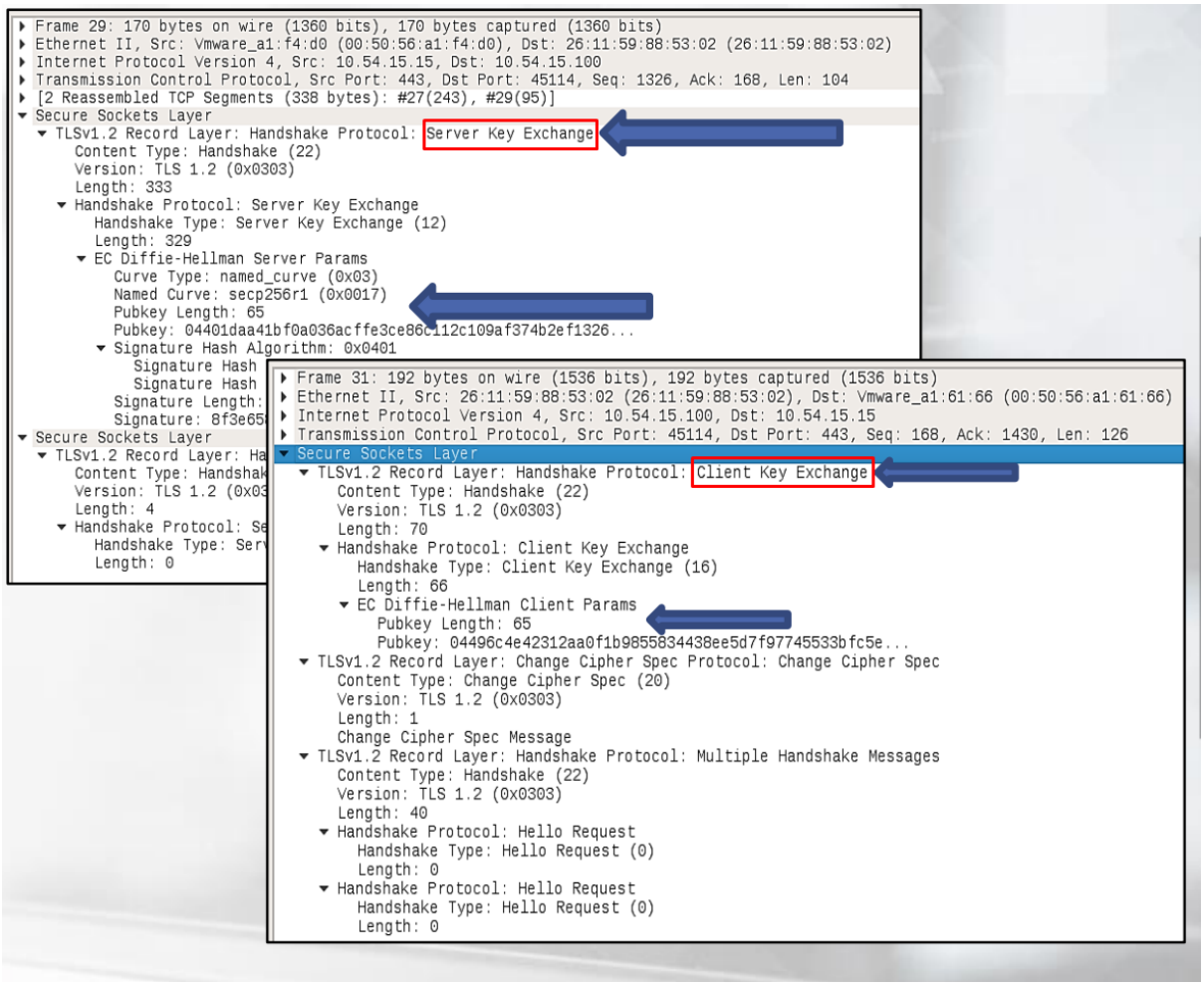
- the Secure Sockets Layer portion of the packet details should not be empty
- in **Client Hello packet**. We see the following:
  - Content Type = Handshak
  - Handshake Protocol: Client Hello
  - Version: TLS 1.2
  - Cipher Suites (11 suites)
  - Compression Method (1 method)



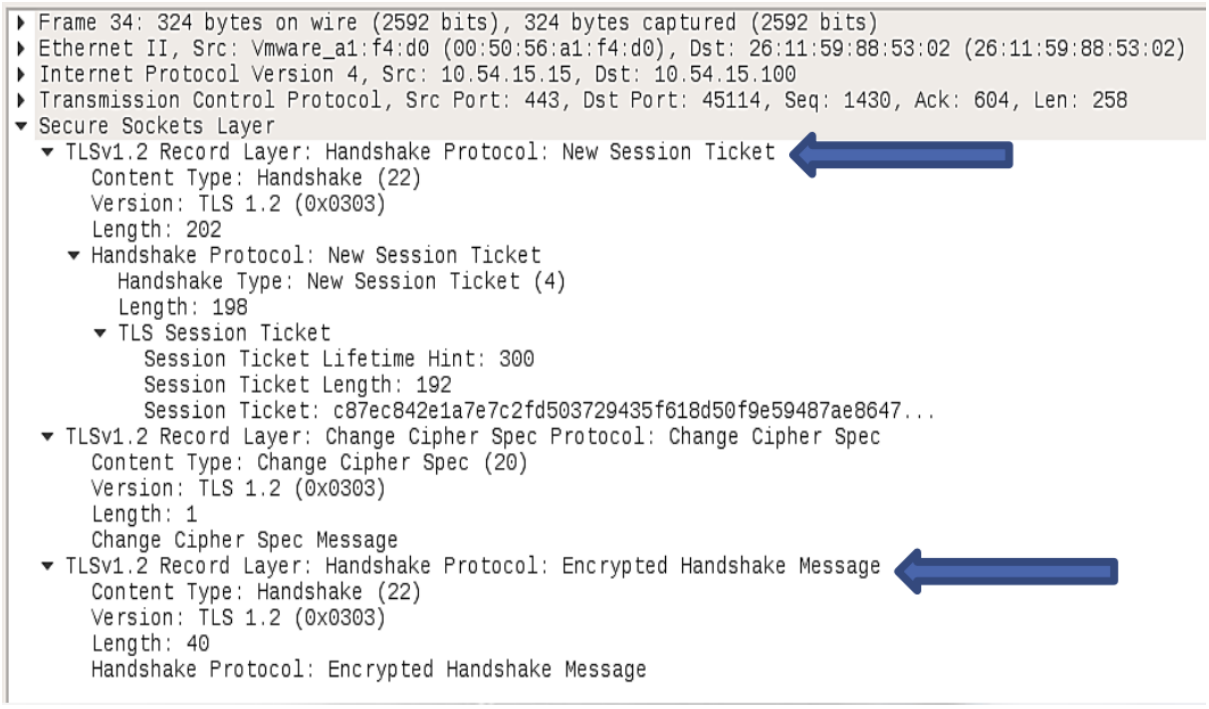
- in the server's response, Server Hello packet



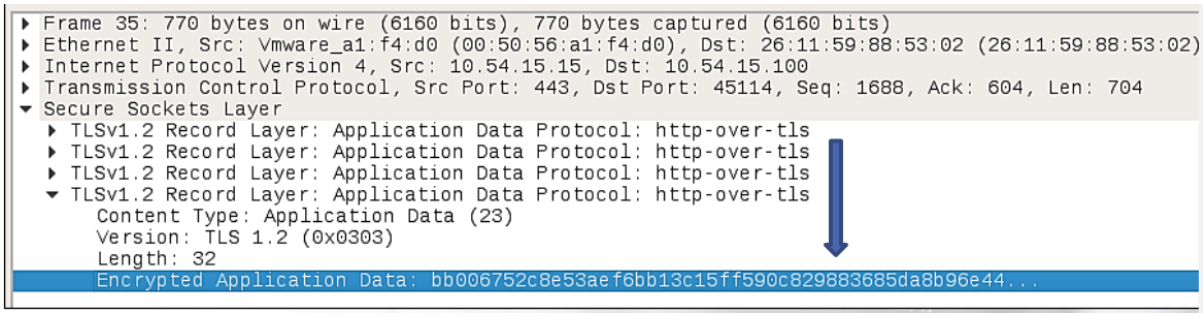
- Here we see the **Server Key Exchange** which will be followed by the **Client Key Exchange** packet. (step #3 in the establishment of an SSL/TLS session)



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

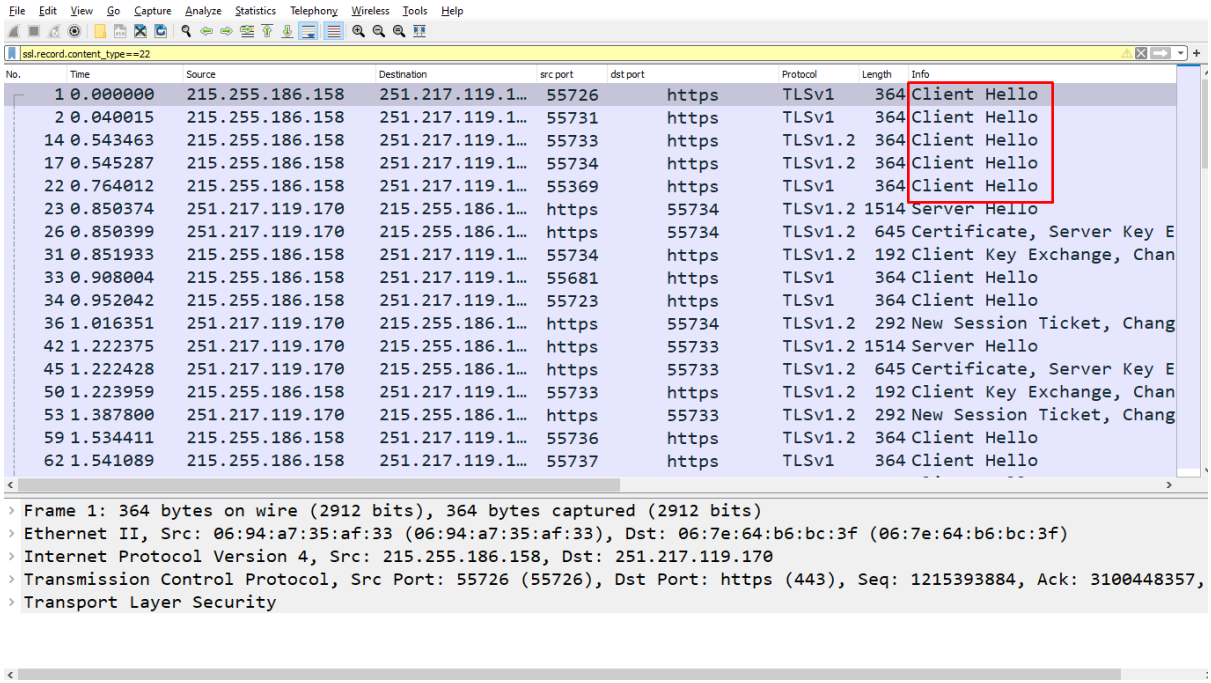


- The rest of the packets between these two devices will now be **encrypted.**
- The traffic is unreadable, but if this is internal traffic within our corporate environment, then, it is feasible to decrypt this traffic using the **private key** from the internal server.



Malicious HTTPS Traffic Example

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



- 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)**

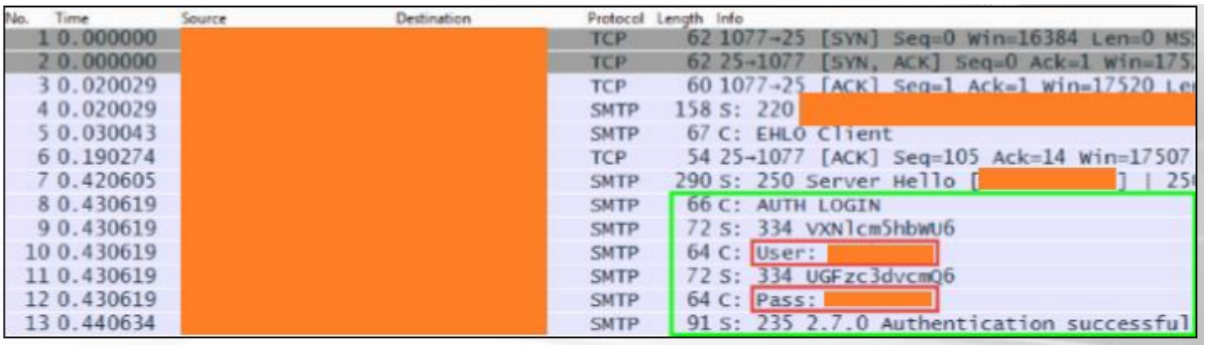
SMTP (Simple Mail Transfer Protocol)

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.



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.

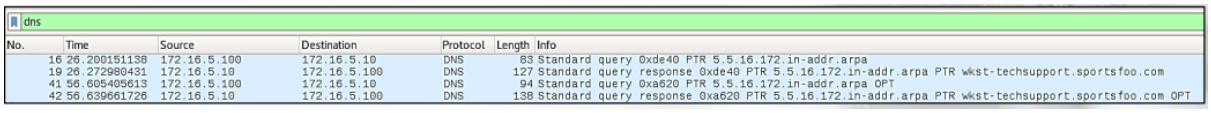
DNS (Domain Name System)

- 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.**

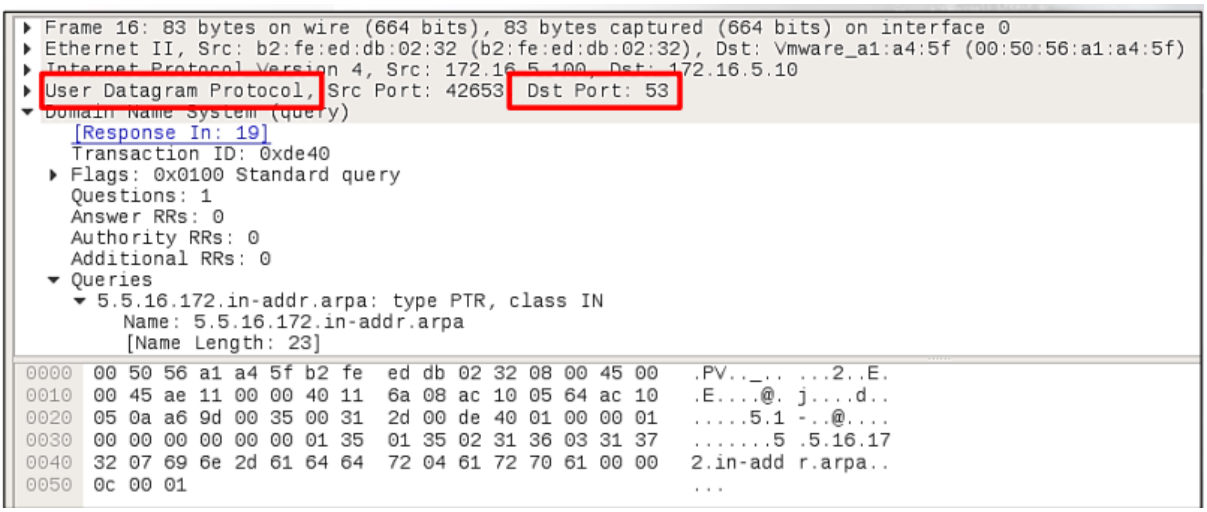
Normal vs Malicious DNS Traffic

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.

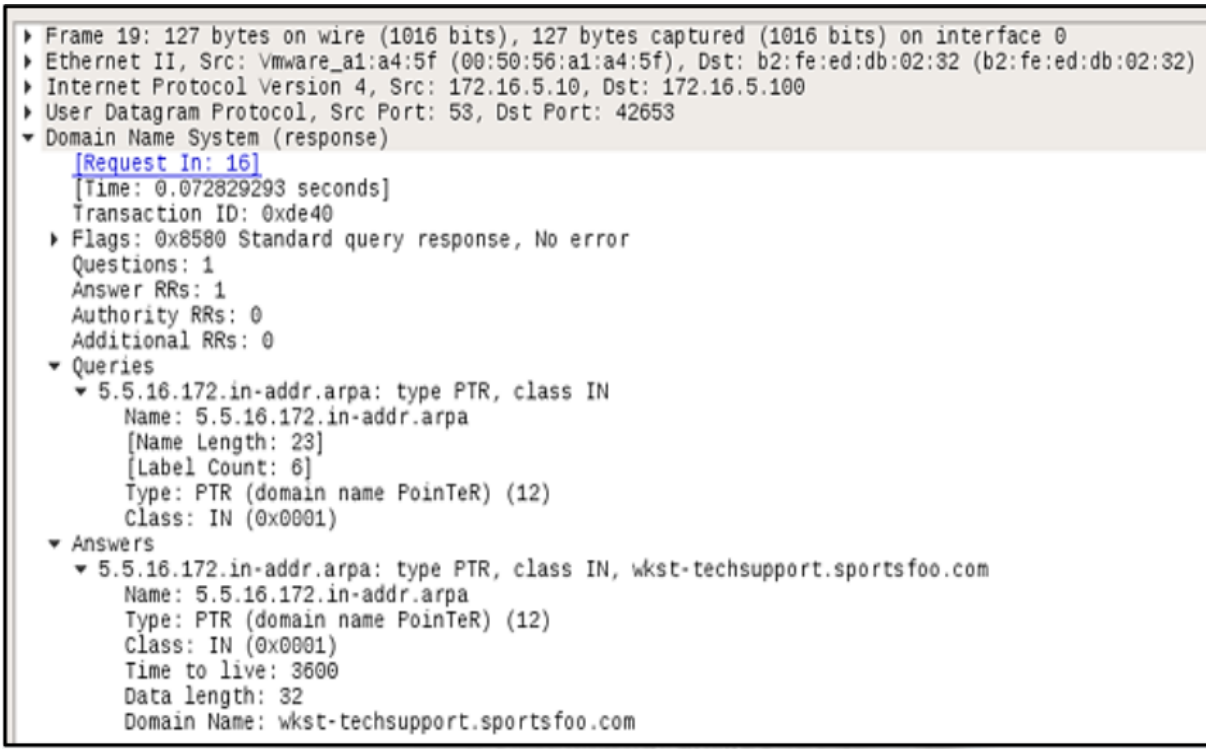
Normal DNS Traffic



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



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



- 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**