TryHackMe Wireshark 101

Saikat Karmakar | AUG 23 : 2021

Collection Methods

Before going into detail about how to analyze each protocol in a PCAP we need to understand the ways to gather a PCAP file. The basic steps to gather a PCAP in Wireshark itself can be simple however bringing into traffic can both the hard part as well as the fun part, this can include: taps, port mirroring, MAC floods, ARP Poisoning. This room will not cover how to set up these various strategies of live packet capturing and will only cover the basic theory of each.

Collection Methods Overview

Some things to think about before going headfirst into attempting to collect and monitor live packet captures.

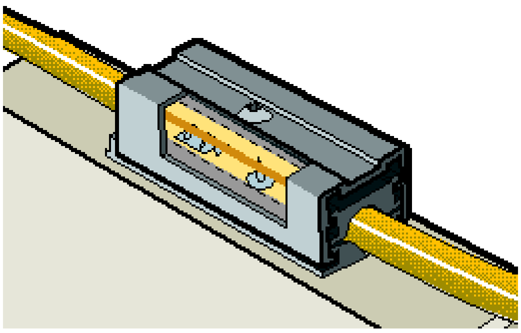
- Begin by starting with a sample capture to ensure that everything is correctly set up and you are successfully capturing traffic.
- Ensure that you have enough compute power to handle the number of packets based on the size of the network, this will obviously vary network by network.
- Ensure enough disk space to store all of the packet captures.

Once you meet all these criteria and have a collection method picked out you can begin to actively monitor and collect packets on a network.

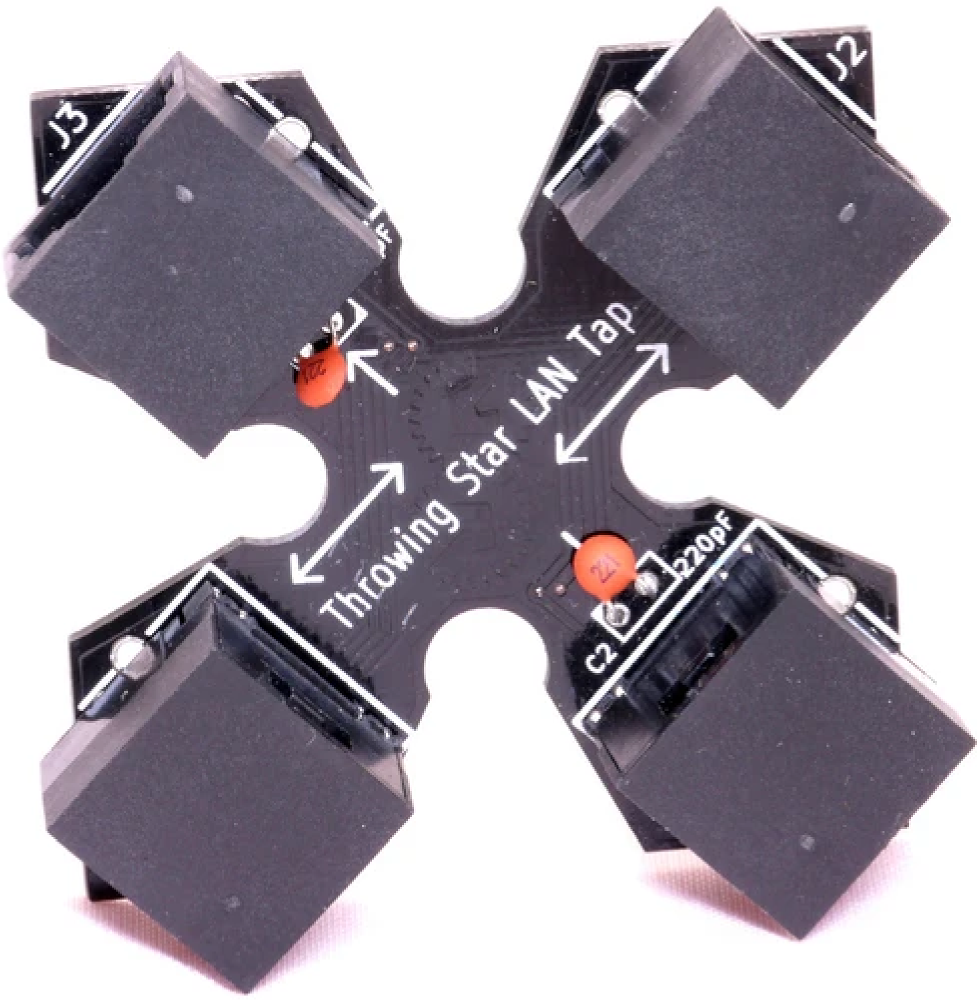
Network Taps

Network taps are a physical implant in which you physically tap between a cable, these techniques are commonly used by Threat Hunting/DFIR teams and red teams in an engagement to sniff and capture packets.

There are two primary means of tapping a wire. The first is by using hardware to tap the wire and intercept the traffic as it comes across, an example of this would be a vampire tap as pictured below.



Another option for planting a network tap would be an inline network tap, which you would plant between or 'inline' two network devices. The tap will replicate packets as they pass the tap. An example of this tap would be the very common Throwing Star LAN Tap



MAC Floods

MAC Floods are a tactic commonly used by red teams as a way of actively sniffing packets. MAC Flooding is intended to stress the switch and fill the CAM table. Once the CAM table is filled the switch will no longer accept new MAC addresses and so in order to keep the network alive, the switch will send out packets to all ports of the switch.

Note: This technique should be used with extreme caution and with explicit prior consent.

ARP Poisoning

ARP Poisoning is another technique used by red teams to actively sniff packets. By ARP Poisoning you can redirect the traffic from the host(s) to the machine you're monitoring from. This technique will not stress network equipment like MAC Flooding however should still be used with caution and only if other techniques like network taps are unavailable.

Filtering Captures

Packet Filtering is a very important part of packet analysis especially when you have a very large number of packet sometimes even 100,000 plus. In task 3 capture filters were briefly covered however there is a second type of filter that is often thought of as more powerful and easier to use. This second method is known as display filters, you can apply display filters in two ways: through the analyze tab and at the filter bar at the top of the packet capture.

Filtering Operators

Wireshark's filter syntax can be simple to understand making it easy to get a hold of quickly. To get the most out of these filters you need to have a basic understanding of boolean and logic operators.

Wireshark only has a few that you will need to be familiar with:

- and - operator: and / &&
- or - operator: or / ||
- equals - operator: eq / ==
- not equal - operator: ne / !=
- greater than - operator: gt / >
- less than - operator: lt / <

Wireshark also has a few other operators that go beyond the power of normal logical operators. These operators are the contains, matches, and bitwise_and operators. These operators can be very useful when you have a large capture and need to pinpoint a single packet. They are out of scope for this room however I recommend doing your own research, the [Wireshark Filtering Documentation] can be a great starting point.

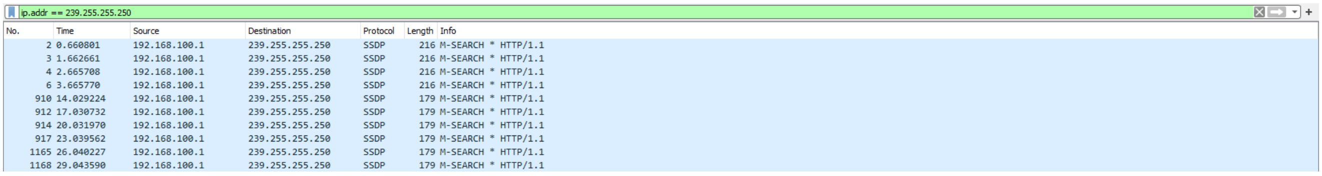
Basic Filtering

Filtering gives us a very large scope of what we can do with the packets, because of this there can be a lot of different filtering syntax options. We will only be covering the very basics in this room such as filtering by IP, protocol, etc. for more information on filtering check out the [Wireshark filtering documentation].

There is a general syntax to the filter commands however they can be a little silly at times. The basic syntax of Wireshark filters is some kind of service or protocol like ip or tcp, followed by a dot then whatever is being filtered for example an address, MAC, SRC, protocol, etc.

Filtering by IP: The first filter we will look at is ip.addr, this filter will allow you to comb through the traffic and only see packets with a specific IP address contained in those packets, whether it be from the source or destination.

Syntax: `ip.addr == <IP Address>`

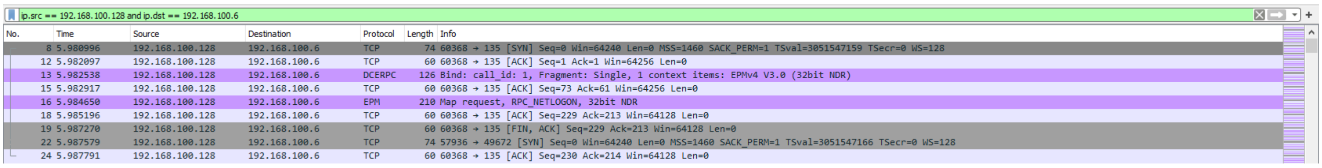

ip.addr == 239.255.255.250

No.	Time	Source	Destination	Protocol	Length	Info
2	0.660801	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
3	1.662661	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
4	2.665708	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
6	3.665770	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
910	14.029224	192.168.100.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
912	17.030732	192.168.100.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
914	20.031970	192.168.100.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
917	23.039562	192.168.100.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
1165	26.040227	192.168.100.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
1168	29.043590	192.168.100.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1

This filter can be handy in practical applications, say when you are threat hunting, and have identified a potentially suspicious host with other tools, you can use Wireshark to further analyze the packets coming from that device.

Filtering by SRC and DST: The second filter will look at is two in one as well as a filter operator: ip.src and ip.dst. These filters allow us to filter the traffic by the source and destination from which the traffic is coming from.

Syntax: `ip.src == <SRC IP Address> and ip.dst == <DST IP Address>`


ip.src == 192.168.100.128 and ip.dst == 192.168.100.6

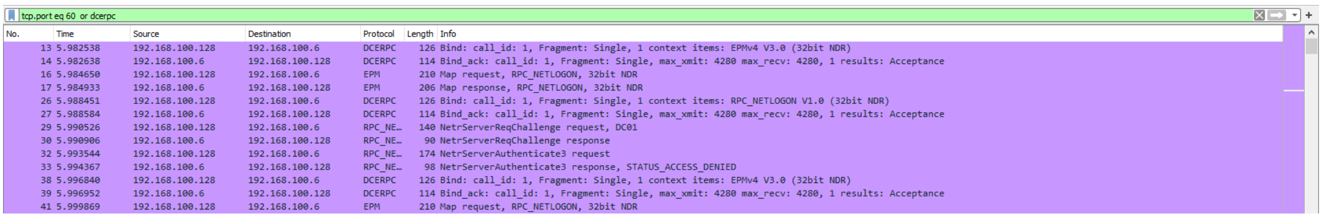
No.	Time	Source	Destination	Protocol	Length	Info
8	5.980996	192.168.100.128	192.168.100.6	TCP	74	60368 → 135 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547159 TSecr=0 WS=128
12	5.982097	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=1 Ack=1 Win=64256 Len=0
13	5.982538	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: EPMv4 V3.0 (32bit NDR)
15	5.982917	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=73 Ack=61 Win=64256 Len=0
16	5.984650	192.168.100.128	192.168.100.6	EPM	210	Map request, RPC_NETLOGON, 32bit NDR
18	5.985196	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=229 Ack=213 Win=64128 Len=0
19	5.987270	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [FIN, ACK] Seq=229 Ack=213 Win=64128 Len=0
22	5.987579	192.168.100.128	192.168.100.6	TCP	74	57936 → 49672 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547166 TSecr=0 WS=128
24	5.987791	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=230 Ack=214 Win=64128 Len=0

Similar to the first filter we can see that Wireshark is combing through the packets and filtering based on the source and destination we set.

Filtering by TCP Protocols: The last filter that we will be covering is the protocol filter, this allows you to set a port or protocol to filter by and can be handy when trying to keep track of an unusual protocol or port being used.

It is worthwhile to mention that Wireshark can filter by both port numbers as well as protocol names.

Syntax: `tcp.port eq <Port #> or <Protocol Name>`


tcp.port eq 60 or dcerpc

No.	Time	Source	Destination	Protocol	Length	Info
13	5.982538	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: EPMv4 V3.0 (32bit NDR)
14	5.982638	192.168.100.6	192.168.100.128	DCERPC	114	Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4280 max_recv: 4280, 1 results: Acceptance
16	5.984650	192.168.100.128	192.168.100.6	EPM	210	Map request, RPC_NETLOGON, 32bit NDR
17	5.984933	192.168.100.6	192.168.100.128	EPM	206	Map response, RPC_NETLOGON, 32bit NDR
26	5.988451	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: RPC_NETLOGON V1.0 (32bit NDR)
27	5.988584	192.168.100.6	192.168.100.128	DCERPC	114	Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4280 max_recv: 4280, 1 results: Acceptance
29	5.990526	192.168.100.128	192.168.100.6	RPC_NE…	140	NetrServerReqChallenge request, DC01
30	5.990906	192.168.100.6	192.168.100.128	RPC_NE…	90	NetrServerReqChallenge response
32	5.993544	192.168.100.128	192.168.100.6	RPC_NE…	174	NetrServerAuthenticate3 request
33	5.994367	192.168.100.6	192.168.100.128	RPC_NE…	98	NetrServerAuthenticate3 response, STATUS_ACCESS_DENIED
38	5.996840	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: EPMv4 V3.0 (32bit NDR)
39	5.996952	192.168.100.6	192.168.100.128	DCERPC	114	Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4280 max_recv: 4280, 1 results: Acceptance
41	5.999869	192.168.100.128	192.168.100.6	EPM	210	Map request, RPC_NETLOGON, 32bit NDR

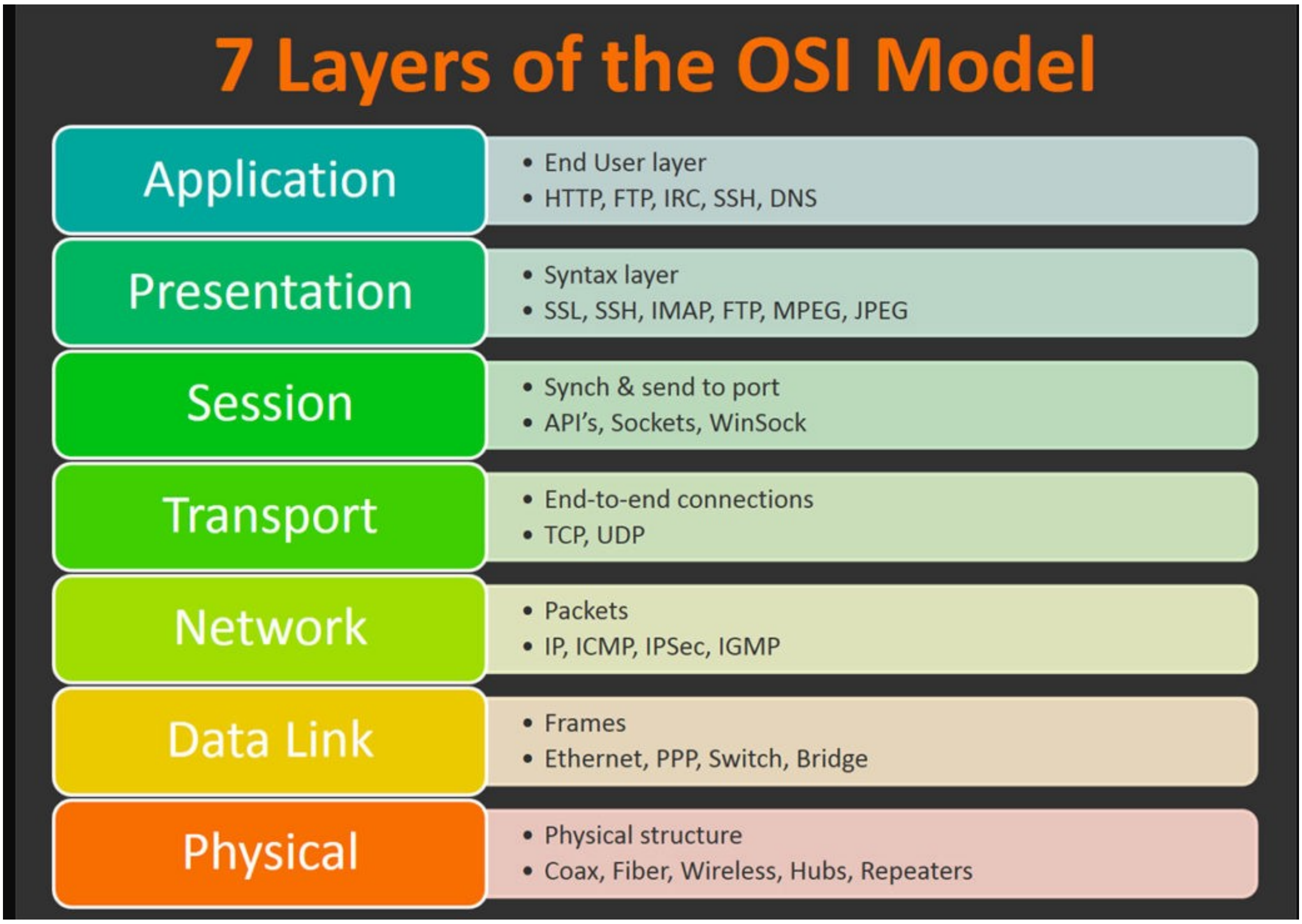
Filtering by UDP Protocols: You can also filter by UDP ports by changing the prefix from tcp to udp

Syntax: `udp.port eq <Port #> or <Protocol Name>`

That is the end of filtering for this task however I recommend you play around with other filters and operators on your own. Once you're ready move on to Task 5.

Packet Dissection

This section covers how Wireshark uses OSI layers to break down packets and how to use these layers for analysis. It is expected that you already have background knowledge of what the OSI model is and how it works.



Raza, M., 2018. 7 Layers Of The OSI Model

Packet Details

You can double click on a packet in capture to open its details. Packets consist of 5 to 7 layers based on the OSI model. We will go over all of them in an HTTP packet from a sample capture.

```
> Frame 27: 214 bytes on wire (1712 bits), 214 bytes captured (1712 bits)
> Ethernet II, Src: fe:ff:20:00:01:00 (fe:ff:20:00:01:00), Dst: Xerox_00:00:00 (00:00:01:00:00:00)
> Internet Protocol Version 4, Src: 216.239.59.99, Dst: 145.254.160.237
> Transmission Control Protocol, Src Port: 80, Dst Port: 3371, Seq: 778787098, Ack: 918692089, Len: 160
> [2 Reassembled TCP Segments (1590 bytes): #26(1430), #27(160)]
> Hypertext Transfer Protocol
> Line-based text data: text/html (3 lines)
```

Looking above we can see 7 distinct layers to the packet: frame/packet, source [MAC], source [IP], protocol, protocol errors, application protocol, and application data. Below we will go over the layers in more detail.

- Frame (Layer 1) -- This will show you what frame / packet you are looking at as well as details specific to the Physical layer of the OSI model.

```
Frame 27: 214 bytes on wire (1712 bits), 214 bytes captured (1712 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: May 13, 2004 06:17:11.266912000 Eastern Daylight Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1084443431.266912000 seconds
    [Time delta from previous captured frame: 0.040058000 seconds]
    [Time delta from previous displayed frame: 0.040058000 seconds]
    [Time since reference or first frame: 3.955688000 seconds]
    Frame Number: 27
    Frame Length: 214 bytes (1712 bits)
    Capture Length: 214 bytes (1712 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:http:data-text-lines]
    [Coloring Rule Name: HTTP]
    [Coloring Rule String: http || tcp.port == 80 || http2]
```

- Source [MAC] (Layer 2) -- This will show you the source and destination MAC Addresses; from the Data Link layer of the OSI model.

```
Ethernet II, Src: fe:ff:20:00:01:00 (fe:ff:20:00:01:00), Dst: Xerox_00:00:00 (00:00:01:00:00:00)
  > Destination: Xerox_00:00:00 (00:00:01:00:00:00)
  > Source: fe:ff:20:00:01:00 (fe:ff:20:00:01:00)
    Type: IPv4 (0x0800)
```

- Source [IP] (Layer 3) -- This will show you the source and destination IPv4 Addresses; from the Network layer of the OSI model.

```
Internet Protocol Version 4, Src: 216.239.59.99, Dst: 145.254.160.237
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
    Total Length: 200
    Identification: 0x85cf (34255)
  > Flags: 0x0000
    Fragment offset: 0
    Time to live: 55
    Protocol: TCP (6)
    Header checksum: 0xb612 [validation disabled]
    [Header checksum status: Unverified]
    Source: 216.239.59.99
    Destination: 145.254.160.237
```

- Protocol (Layer 4) -- This will show you details of the protocol used (UDP/TCP) along with source and destination ports; from the Transport layer of the OSI model.

```
Transmission Control Protocol, Src Port: 80, Dst Port: 3371, Seq: 778787098, Ack: 918692089, Len: 160
    Source Port: 80
    Destination Port: 3371
    [Stream index: 1]
    [TCP Segment Len: 160]
    Sequence number: 778787098
    [Next sequence number: 778787258]
    Acknowledgment number: 918692089
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 31460
    [Calculated window size: 31460]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xde29 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
    TCP payload (160 bytes)
    TCP segment data (160 bytes)
```

- Protocol Errors -- This is a continuation of the 4th layer showing specific segments from TCP that needed to be reassembled.

```
[2 Reassembled TCP Segments (1590 bytes): #26(1430), #27(160)]
    [Frame: 26, payload: 0-1429 (1430 bytes)]
    [Frame: 27, payload: 1430-1589 (160 bytes)]
    [Segment count: 2]
    [Reassembled TCP length: 1590]
    [Reassembled TCP Data: 485454502f312e3120323030204f4b0d0a5033503a20706f…]
```

- Application Protocol (Layer 5) -- This will show details specific to the protocol being used such HTTP, FTP, SMB, etc. From the Application layer of the OSI model.

```
Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    P3P: policyref="http://www.googleadservices.com/pagead/p3p.xml", CP="NOI DEV PSA PSD IVA PVD OTP OUR OTR IND OTC"\r\n
    Content-Type: text/html; charset=ISO-8859-1\r\n
    Content-Encoding: gzip\r\n
    Server: CAFE/1.0\r\n
    Cache-control: private, x-gzip-ok=""\r\n
  > Content-length: 1272\r\n
    Date: Thu, 13 May 2004 10:17:14 GMT\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.971397000 seconds]
    [Request in frame: 18]
    [Request URI [truncated]: http://pagead2.googlesyndication.com/pagead/ads?client=ca-pub-2309191948673629&random=108444
    Content-encoded entity body (gzip): 1272 bytes -> 3608 bytes
    File Data: 3608 bytes
```

- Application Data -- This is an extension of layer 5 that can show the application-specific data.

```
Line-based text data: text/html (3 lines)
    <html><head><style><!--\n
     [truncated].ch{cursor:pointer;cursor:hand}a.ad:link { color: #000000 }
     [truncated]function ss(w,id) {window.status = w;return true;}function
```

ARP Traffic

ARP Overview

ARP or Address Resolution Protocol is a Layer 2 protocol that is used to connect IP Addresses with MAC Addresses. They will contain REQUEST messages and RESPONSE messages. To identify packets the message header will contain one of two operation codes:

- Request (1)
- Reply (2)

Below you can see a packet capture of multiple ARP requests and replies.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Intel_78:0c:02	Broadcast	ARP	60	Who has 192.168.1.1? Tell 192.168.1.3
3	0.017234	ThomsonT_eb:46:e7	Intel_78:0c:02	ARP	42	192.168.1.1 is at 00:90:d0:eb:46:e7
5	0.096040	Intel_78:0c:02	Broadcast	ARP	82	Who has 192.168.1.1? Tell 192.168.1.3
11	25.478711	Intel_78:0c:02	Broadcast	ARP	60	Who has 192.168.1.2? Tell 192.168.1.3
12	25.491556	Intel_78:0c:02	Broadcast	ARP	82	Who has 192.168.1.2? Tell 192.168.1.3
13	25.492485	CompexUs_24:33:32	Intel_78:0c:02	ARP	82	192.168.1.2 is at 00:80:48:24:33:32
15	25.493377	CompexUs_24:33:32	Intel_78:0c:02	ARP	42	192.168.1.2 is at 00:80:48:24:33:32

It is useful to note that most devices will identify themselves or Wireshark will identify it such as Intel_78, an example of suspicious traffic would be many requests from an unrecognized source. You need to enable a setting within Wireshark however to resolve physical addresses. To enable this feature, navigate to View > Name Resolution > Ensure that Resolve Physical Addresses is checked.

Looking at the below screenshot we can see that a Cisco device is sending ARP Requests, meaning that we should be able to trust this device, however you should always stay on the side of caution when analyzing packets.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Cisco251_af:f4:54	Broadcast	ARP	60	Who has 24.166.173.159? Tell 24.166.172.1
2	0.098594	Cisco251_af:f4:54	Broadcast	ARP	60	Who has 24.166.172.141? Tell 24.166.172.1
3	0.110617	Cisco251_af:f4:54	Broadcast	ARP	60	Who has 24.166.173.161? Tell 24.166.172.1
4	0.211791	Cisco251_af:f4:54	Broadcast	ARP	60	Who has 65.28.78.76? Tell 65.28.78.1
5	0.216744	Cisco251_af:f4:54	Broadcast	ARP	60	Who has 24.166.173.163? Tell 24.166.172.1
6	0.307909	Cisco251_af:f4:54	Broadcast	ARP	60	Who has 24.166.175.123? Tell 24.166.172.1
7	0.330433	Cisco251_af:f4:54	Broadcast	ARP	60	Who has 24.166.173.165? Tell 24.166.172.1

ARP Traffic Overview

ARP Request Packets:

We can begin analyzing packets by looking at the first ARP Request packet and looking at the packet details.

Wireshark · Packet 1 · dns-remoteshell.pcap

```
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
> Ethernet II, Src: Intel_78:0c:02 (00:0e:35:78:0c:02), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 00:0e:35:78:0c:02
    Sender IP address: 192.168.1.3
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.1.1
```

Looking at the packet details above, the most important details of the packet are outlined in red. The Opcode is short for operation code and will you tell you whether it is an ARP Request or Reply. The second outlined detail is to where the packet is requesting to, in this case, it is broadcasting the request to all.

ARP Reply Packets:

Wireshark · Packet 3 · dns-remoteshell.pcap

```
> Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
> Ethernet II, Src: 00:90:d0:eb:46:e7, Dst: 00:0e:35:78:0c:02
v Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 00:90:d0:eb:46:e7
    Sender IP address: 192.168.1.1
    Target MAC address: 00:0e:35:78:0c:02
    Target IP address: 192.168.1.3
```

Looking at the above packet details we can see from the Opcode that it is an ARP Reply packet. We can also get other useful information like the MAC and IP Address that was sent along with the reply since this is a reply packet we know that this was the information sent along with the message.

ARP is one of the simpler protocols to analyze, all you need to remember is to identify whether it is a request or reply packet and who it is being sent by.

Practical ARP Packet Analysis

Now that you know what ARP packets and normal traffic look like download the provided PCAP or *nb6-startup.pcap* from the [Wireshark website](). This capture has multiple protocols so you may need to use your knowledge of filtering from previous tasks; once you're ready, begin analysis of the capture.

ICMP Traffic

ICMP Overview

ICMP or Internet Control Message Protocol is used to analyze various nodes on a network. This is most commonly used with utilities like ping and traceroute. You should already be familiar with how ICMP works; however, if you need a refresher, read the [IETF documentation]().

Below you can see a sample of what a ping would look like, we can see a request to the server from ICMP, then a reply from the server.

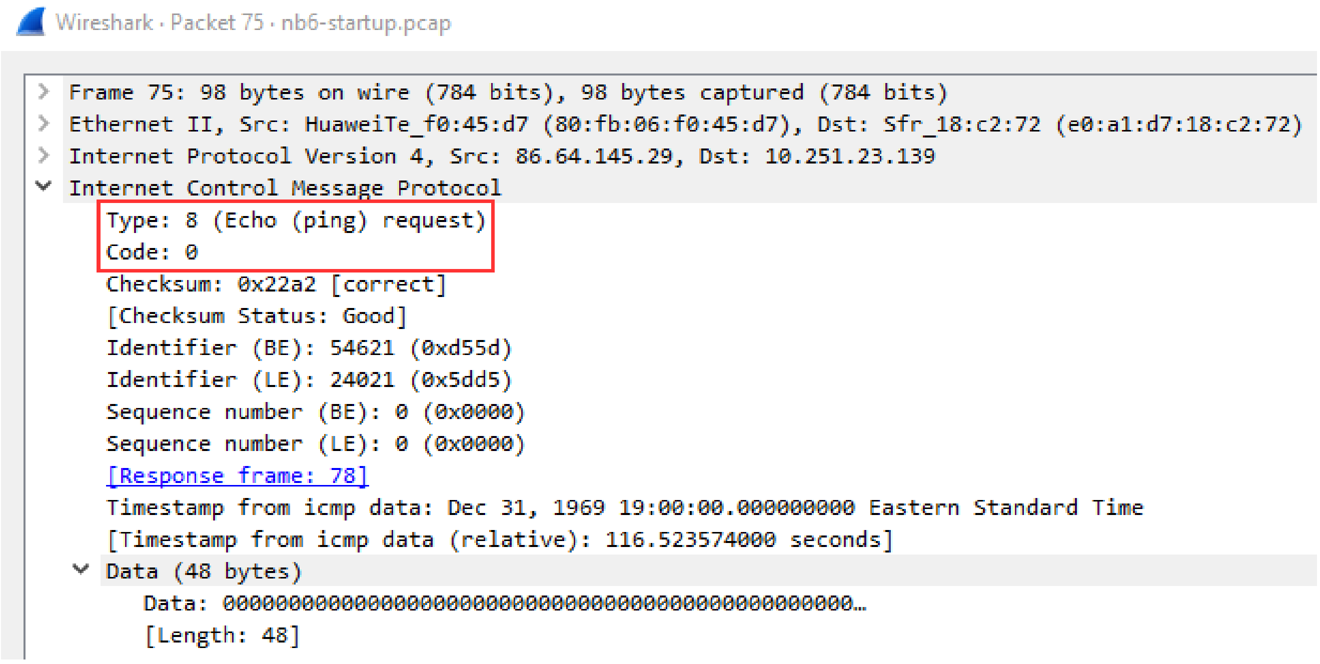
No.	Time	Source	Destination	Protocol	Length	Info
75	61.879584	86.64.145.29	10.251.23.139	ICMP	98	Echo (ping) request id=0xd55d, seq=0/0, ttl=59 (reply in 78)
78	61.879932	10.251.23.139	86.64.145.29	ICMP	98	Echo (ping) reply id=0xd55d, seq=0/0, ttl=64 (request in 75)

ICMP Traffic Overview

ICMP request:

Below we see packet details for a ping request packet. There are a few important things within the packet details that we can take note of first being the type and code of the packet. A type that equals 8 means that it is a request packet, if it is equal to 0 it is a reply packet. When these codes are altered or do not seem correct that is typically a sign of suspicious activity.

There are two other details within the packet that are useful to analyze: timestamp and data. The timestamp can be useful for identifying the time the ping was requested it can also be useful to identify suspicious activity in some cases. We can also look at the data string which will typically just be a random data string.

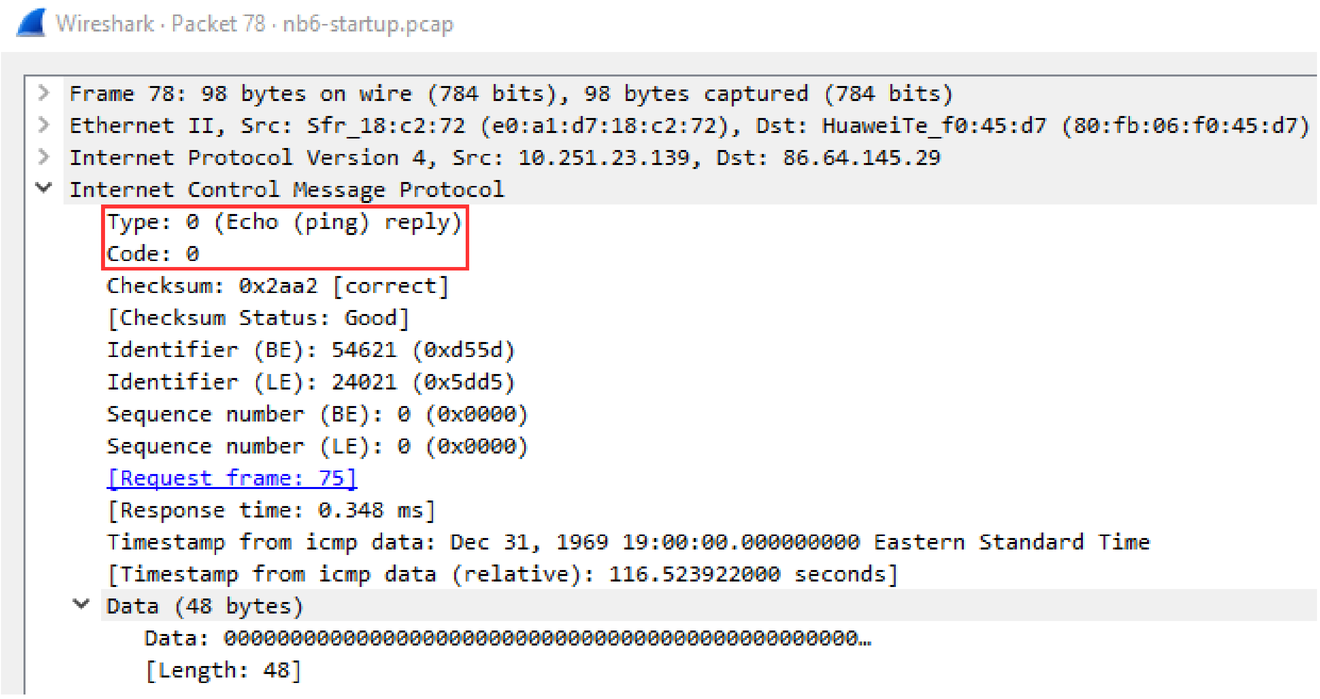


```
Wireshark · Packet 75 · nb6-startup.pcap
> Frame 75: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: HuaweiTe_f0:45:d7 (80:fb:06:f0:45:d7), Dst: Sfr_18:c2:72 (e0:a1:d7:18:c2:72)
> Internet Protocol Version 4, Src: 86.64.145.29, Dst: 10.251.23.139
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x22a2 [correct]
    [Checksum Status: Good]
    Identifier (BE): 54621 (0xd55d)
    Identifier (LE): 24021 (0x5dd5)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Response frame: 78]
    Timestamp from icmp data: Dec 31, 1969 19:00:00.000000000 Eastern Standard Time
    [Timestamp from icmp data (relative): 116.523574000 seconds]
  v Data (48 bytes)
      Data: 000000000000000000000000000000000000000000000000…
      [Length: 48]
```

ICMP Reply:

Below you can see that the reply packet is very similar to the request packet. One of the main difference that distinguishes a reply packet is the code, in this case, you can see it is 0, confirming that it is a reply packet.

The same analysis techniques for Request packets apply here as well, again the main difference will be the packet type.



```
Wireshark · Packet 78 · nb6-startup.pcap
> Frame 78: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: Sfr_18:c2:72 (e0:a1:d7:18:c2:72), Dst: HuaweiTe_f0:45:d7 (80:fb:06:f0:45:d7)
> Internet Protocol Version 4, Src: 10.251.23.139, Dst: 86.64.145.29
v Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x2aa2 [correct]
    [Checksum Status: Good]
    Identifier (BE): 54621 (0xd55d)
    Identifier (LE): 24021 (0x5dd5)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Request frame: 75]
    [Response time: 0.348 ms]
    Timestamp from icmp data: Dec 31, 1969 19:00:00.000000000 Eastern Standard Time
    [Timestamp from icmp data (relative): 116.523922000 seconds]
  v Data (48 bytes)
      Data: 000000000000000000000000000000000000000000000000…
      [Length: 48]
```

Practical ICMP Packet Analysis

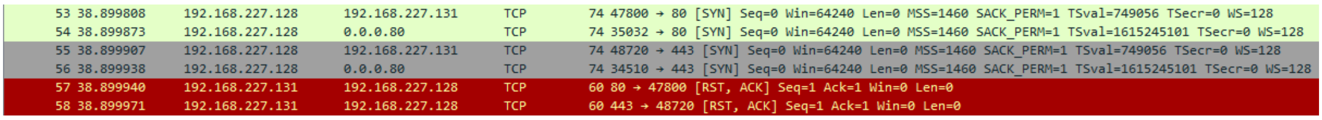
Now that you understand how an ICMP packet is formed and what it contains, we can begin hands-on practical analysis of ICMP packets. Download the provided PCAP or dns+icmp.pcapng.gz from the Wireshark website. This capture only has two protocols so it is up to you whether or not you decide to filter the ICMP protocol or not.

TCP Traffic

TCP Overview

TCP or Transmission Control Protocol handles the delivery of packets including sequencing and errors. You should already have an understanding of how TCP works, if you need a refresher check out the IETF TCP Documentation.

Below you can see a sample of a Nmap scan, scanning port 80 and 443. We can tell that the port is closed due to the RST, ACK packet in red.



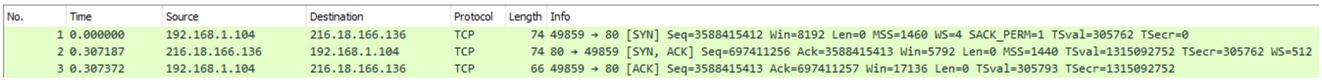
53	38.899808	192.168.227.128	192.168.227.131	TCP	74 47800 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=749056 TSecr=0 WS=128
54	38.899873	192.168.227.128	0.0.0.80	TCP	74 35032 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1615245101 TSecr=0 WS=128
55	38.899907	192.168.227.128	192.168.227.131	TCP	74 48720 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=749056 TSecr=0 WS=128
56	38.899938	192.168.227.128	0.0.0.80	TCP	74 34510 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1615245101 TSecr=0 WS=128
57	38.899940	192.168.227.131	192.168.227.128	TCP	60 80 → 47800 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
58	38.899971	192.168.227.131	192.168.227.128	TCP	60 443 → 48720 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

When analyzing TCP packets, Wireshark can be very helpful and color code the packets in order of danger level. If you can't remember the color code go back to Task 3 and refresh on how Wireshark uses colors to match packets.

TCP can give useful insight into a network when analyzing however it can also be hard to analyze due to the number of packets it sends. This is where you may need to use other tools like RSA NetWitness and NetworkMiner to filter out and further analyze the captures.

TCP Traffic Overview

A common thing that you will see when analyzing TCP packets is known as the TCP handshake, which you should already be familiar with. It includes a series of packets: syn, synack, ack; That allows devices to establish a connection.



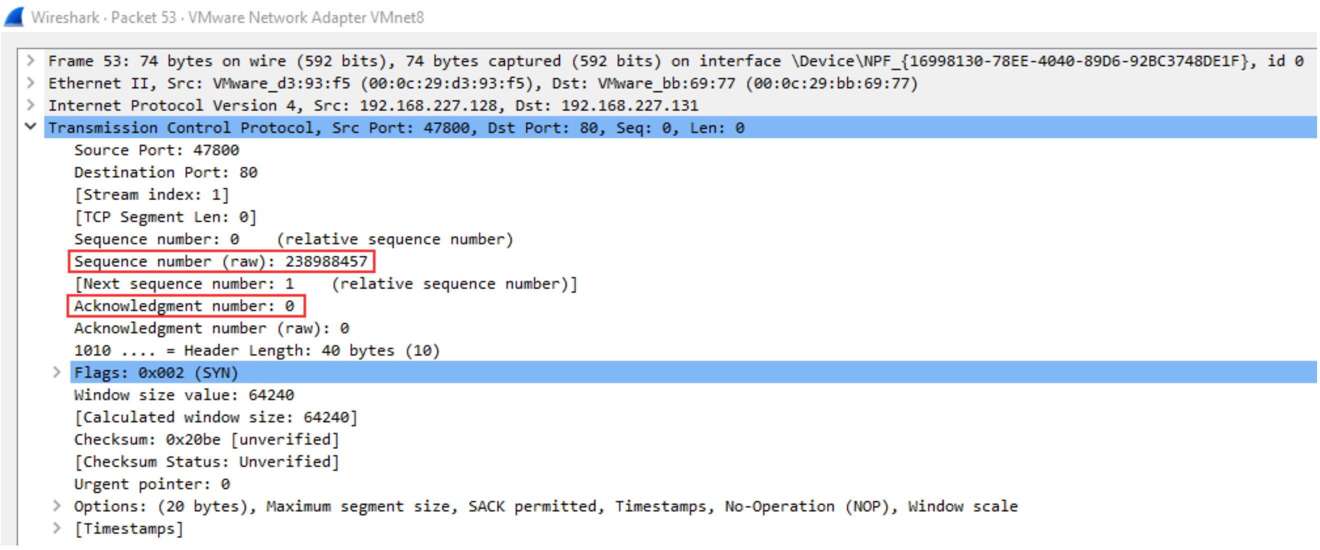
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.104	216.18.166.136	TCP	74	49859 → 80 [SYN] Seq=3588415412 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1 TSval=305762 TSecr=0
2	0.307187	216.18.166.136	192.168.1.104	TCP	74	80 → 49859 [SYN, ACK] Seq=697411256 Ack=3588415413 Win=5792 Len=0 MSS=1440 TSval=1315092752 TSecr=305762 WS=512
3	0.307372	192.168.1.104	216.18.166.136	TCP	66	49859 → 80 [ACK] Seq=3588415413 Ack=697411257 Win=17136 Len=0 TSval=305793 TSecr=1315092752

Typically when this handshake is out of order or when it includes other packets like an RST packet, something suspicious or wrong is happening in the network. The Nmap scan in the section above is a perfect example of this.

TCP Packet Analysis

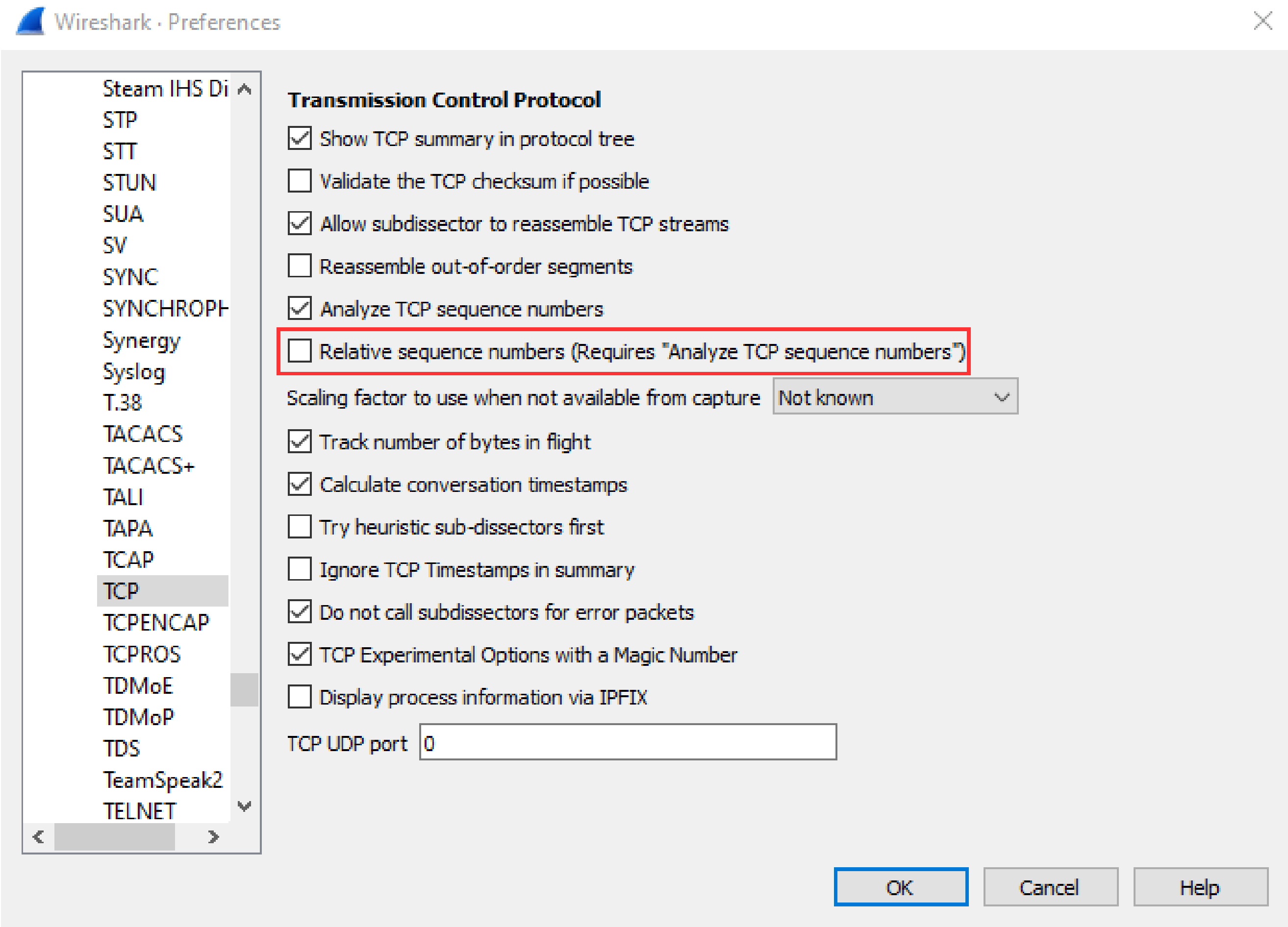
For analyzing TCP packets we will not go into the details of each individual detail of the packets; however, look at a few of the behaviors and structures that the packets have.

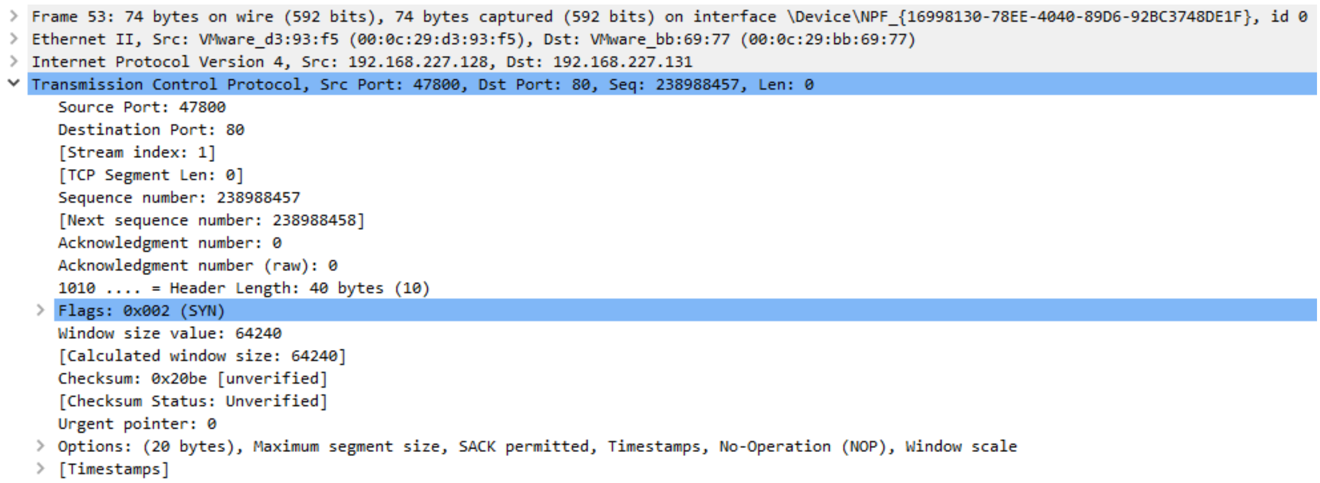
Below we see packet details for an SYN packet. The main thing that we want to look for when looking at a TCP packet is the sequence number and acknowledgment number.



In this case, we see that the port was not open because the acknowledgment number is 0.

Within Wireshark, we can also see the original sequence number by navigating to edit > preferences > protocols > TCP > relative sequence numbers (uncheck boxes).





Typically TCP packets need to be looked at as a whole to tell a story rather than one by one at the details.

DNS Traffic

DNS Overview

DNS or Domain Name Service protocol is used to resolves names with IP addresses. Just like the other protocols, you should be familiar with DNS; however, if you're not you can refresh with the IETF DNS Documentation.

There are a couple of things outlined below that you should keep in the back of your mind when analyzing DNS packets.

- Query-Response
- DNS-Servers Only
- UDP

If anyone of these is out of place then the packets should be looked at further and should be considered suspicious.

Below we can see a packet capture with multiple DNS queries and responses.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.43.9	192.168.43.1	DNS	80	Standard query 0x528e PTR 8.8.8.8.in-addr.arpa
2	5.001009	192.168.43.9	192.168.43.1	DNS	80	Standard query 0x528e PTR 8.8.8.8.in-addr.arpa
3	5.006792	192.168.43.1	192.168.43.9	DNS	124	Standard query response 0x528e PTR 8.8.8.8.in-addr.arpa PTR google-public-dns-a.google.com
10	7.791410	192.168.43.9	192.168.43.1	DNS	80	Standard query 0x695d PTR 4.4.8.8.in-addr.arpa
11	7.979359	192.168.43.1	192.168.43.9	DNS	124	Standard query response 0x695d PTR 4.4.8.8.in-addr.arpa PTR google-public-dns-b.google.com
16	11.999365	192.168.43.9	192.168.43.1	DNS	80	Standard query 0x833a PTR 2.2.2.4.in-addr.arpa
17	12.073341	192.168.43.1	192.168.43.9	DNS	116	Standard query response 0x833a PTR 2.2.2.4.in-addr.arpa PTR b.resolvers.Level3.net

Instantly looking at the packets we can see what they are querying, this can be useful when you have many packets and need to identify suspicious or unusual traffic quickly.

DNS Traffic Overview

DNS Query:

Looking at the below query we really have two bits of information that we can use to analyze the packet. The first bit of information we can look at is where the query is originating from, in this case, it is UDP 53 which means that this packet passes that check, if it was TCP 53 then it should be considered suspicious traffic and needs to analyzed further. We can also look at what it is querying as well, this can be useful with other information to build a story of what happened.

Wireshark · Packet 1 · dns+icmp.pcapng

```
> Frame 1: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface en1, id 0
> Ethernet II, Src: Apple_13:c5:58 (60:33:4b:13:c5:58), Dst: MS-NLB-PhysServer-26_11:f0:c8:3b (02:1a:11:f0:c8:3b)
> Internet Protocol Version 4, Src: 192.168.43.9, Dst: 192.168.43.1
> User Datagram Protocol, Src Port: 51677, Dst Port: 53
v Domain Name System (query)
    Transaction ID: 0x528e
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > 8.8.8.8.in-addr.arpa: type PTR, class IN
```

When analyzing DNS packets you really need to understand your environment and whether or not the traffic would be considered normal within your environment.

DNS Response:

Below we see a response packet, it is similar to the query packet, but it includes an answer as well which can be used to verify the query.

Wireshark · Packet 3 · dns+icmp.pcapng

```
> Frame 3: 124 bytes on wire (992 bits), 124 bytes captured (992 bits) on interface en1, id 0
> Ethernet II, Src: MS-NLB-PhysServer-26_11:f0:c8:3b (02:1a:11:f0:c8:3b), Dst: Apple_13:c5:58 (60:33:4b:13:c5:58)
> Internet Protocol Version 4, Src: 192.168.43.1, Dst: 192.168.43.9
> User Datagram Protocol, Src Port: 53, Dst Port: 51677
v Domain Name System (response)
    Transaction ID: 0x528e
  > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > 8.8.8.8.in-addr.arpa: type PTR, class IN
  v Answers
    > 8.8.8.8.in-addr.arpa: type PTR, class IN, google-public-dns-a.google.com
    [Request In: 2]
    [Time: 0.005783000 seconds]
```

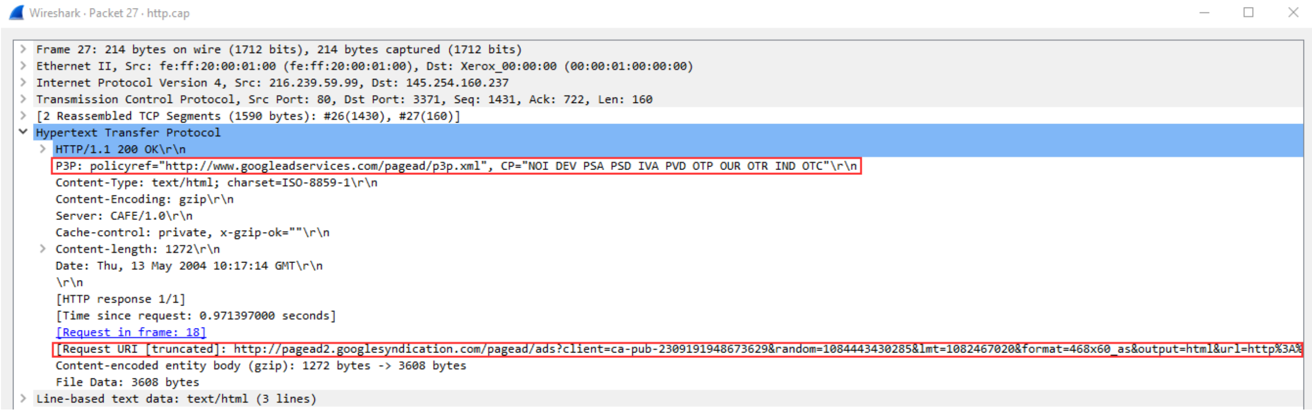
HTTP Traffic

HTTP or Hypertext Transfer Protocol is a commonly used port for the world wide web and used by some websites, however, its encrypted counterpart: HTTPS is more common which we will discuss in the next text. HTTP is used to send GET and POST requests to a web server in order to receive things like webpages. Knowing how to analyze HTTP can be helpful to quickly spot things like SQLi, Web Shells, and other web-related attack vectors.

HTTP Traffic Overview

You should already have a general understanding of how HTTP works before completing this room; however, if you need a refresher you can read the official paper by the [IETF on HTTP methods](#).

HTTP is one of the most straight forward protocols for packet analysis, the protocol is straight to the point and does not include any handshakes or prerequisites before communication.

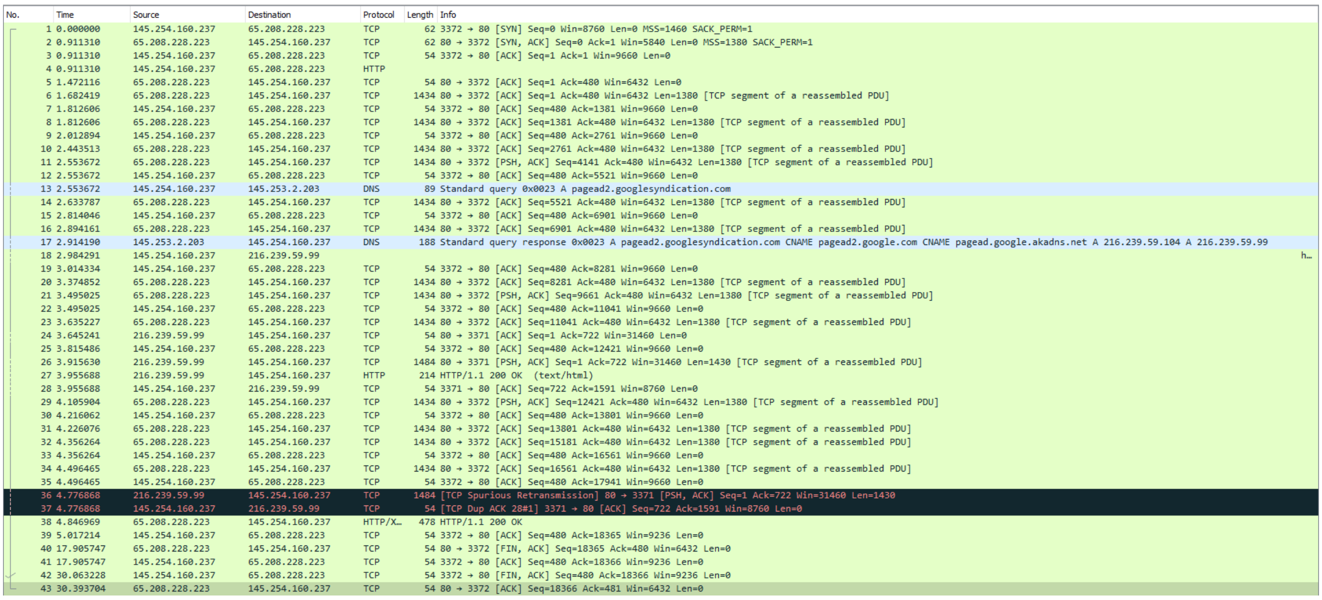


Above we can see a sample HTTP packet, looking at an HTTP packet we can easily gather information since the data stream is not encrypted like the HTTP counterpart HTTPS. Some of the important information we can gather from the packet is the Request URI, File Data, Server.

Now that we understand the basic structure of an HTTP packet we can move on to looking at a sample HTTP packet capture to get hands-on with the packets.

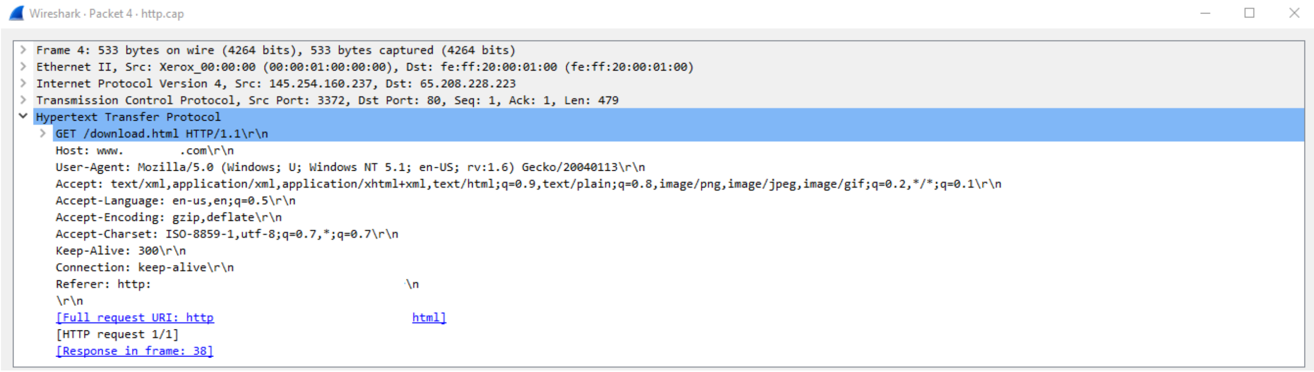
Practical HTTP Packet Analysis

To get an understanding of the flow of HTTP packets and get hands-on with the packets, we can analyze the http.cap file provided by Wireshark. Download the needed file from the task or directly from the [Wireshark website](#).



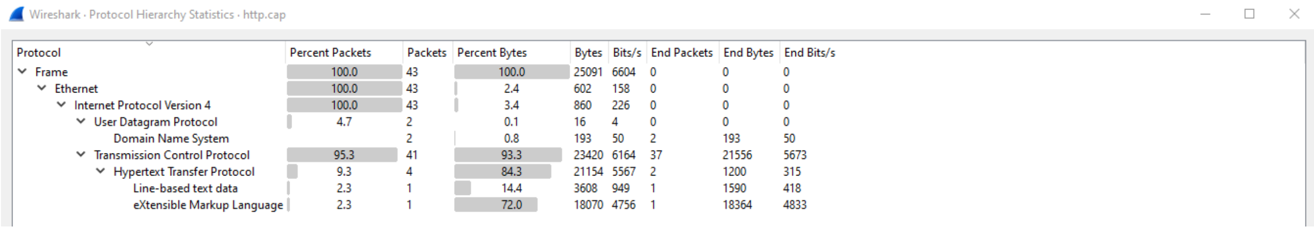
After opening the PCAP we can see that this is just a simple HTTP packet capture with a few requests.

Navigating deeper into the packet capture we can look at the details of one of the HTTP requests for example packet 4.



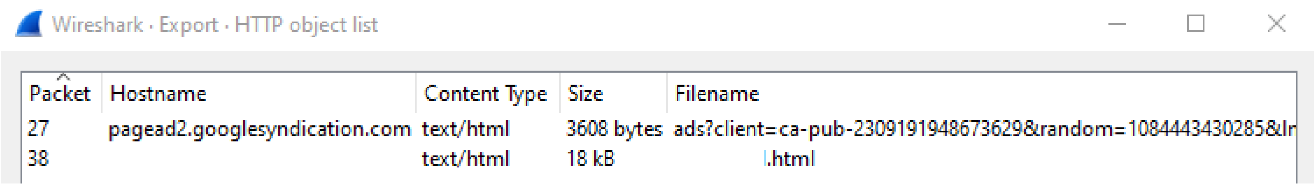
From this packet we can identify some very important information like the host, user-agent, requested URI, and response.

We can use some of Wireshark's built-in features to help digest all of this data and organize it for further future analysis. We can begin by looking at a very useful feature in Wireshark to organize the protocols present in a capture the Protocol Hierarchy. Navigate to Statistics > Protocol Hierarchy.



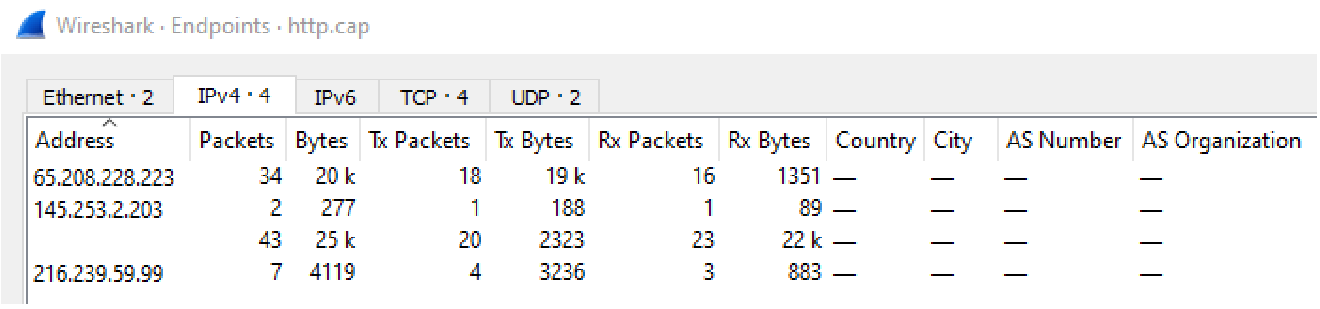
This information can be very useful in practical applications like threat hunting to identify discrepancies in packet captures.

The next feature in Wireshark we will look at is the Export HTTP Object. This feature will allow us to organize all requested URIs in the capture. To use Export HTTP Object navigate to file > Export Objects > HTTP.



Similar to the Protocol Hierarchy this can be useful to quickly identify possible discrepancies in captures.

The last feature we will cover in this section of this room is Endpoints. This feature allows the user to organize all endpoints and IPs found within a specific capture. Just like the other features, this can be useful to identify where a discrepancy is originating from. To use the Endpoints feature navigate to Statistics > Endpoints.



HTTP is not a common protocol to see too much as HTTPS is now more commonly used; however, HTTP is still used often and can be very easy to analyze if given the opportunity.

HTTPS Traffic

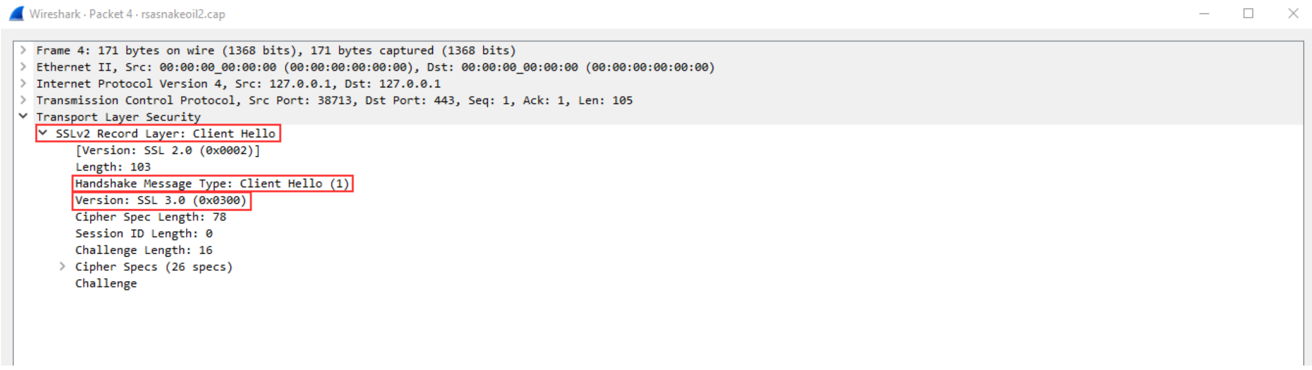
HTTPS or Hypertext Transfer Protocol Secure can be one of the most annoying protocols to understand from a packet analysis perspective and can be confusing to understand the steps needed to take in order to analyze HTTPS packets.

HTTPS Traffic Overview

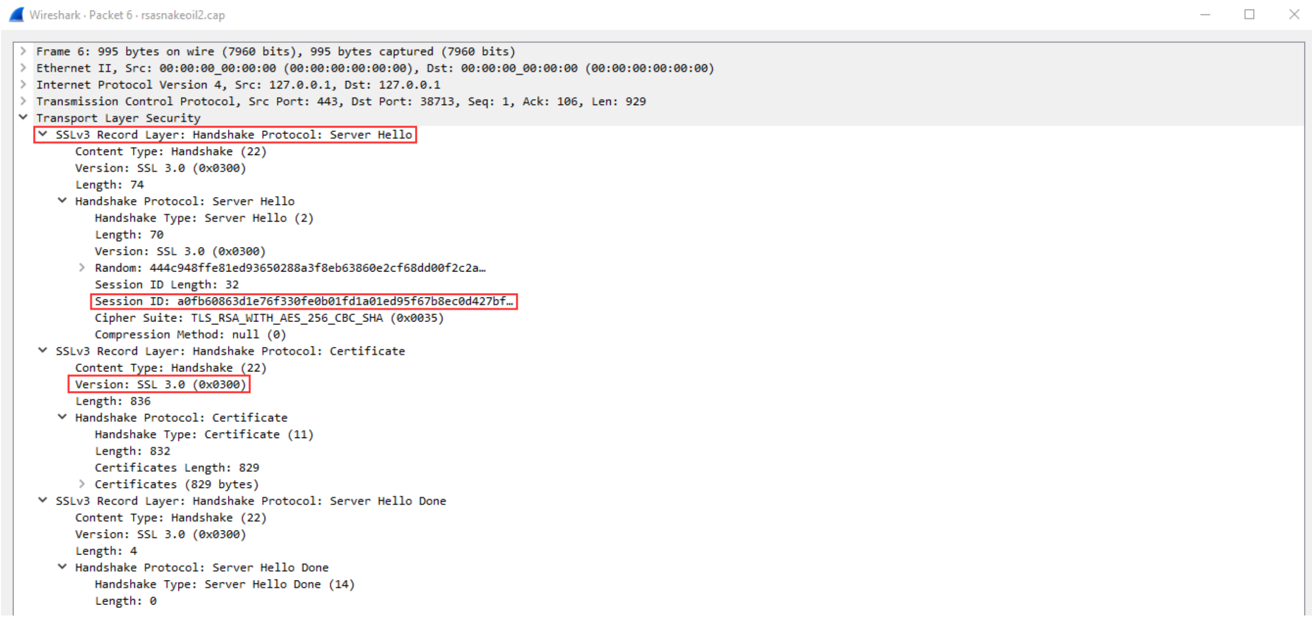
Before sending encrypted information the client and server need to agree upon various steps in order to make a secure tunnel.

1. Client and server agree on a protocol version
2. Client and server select a cryptographic algorithm
3. The client and server can authenticate to each other; this step is optional
4. Creates a secure tunnel with a public key

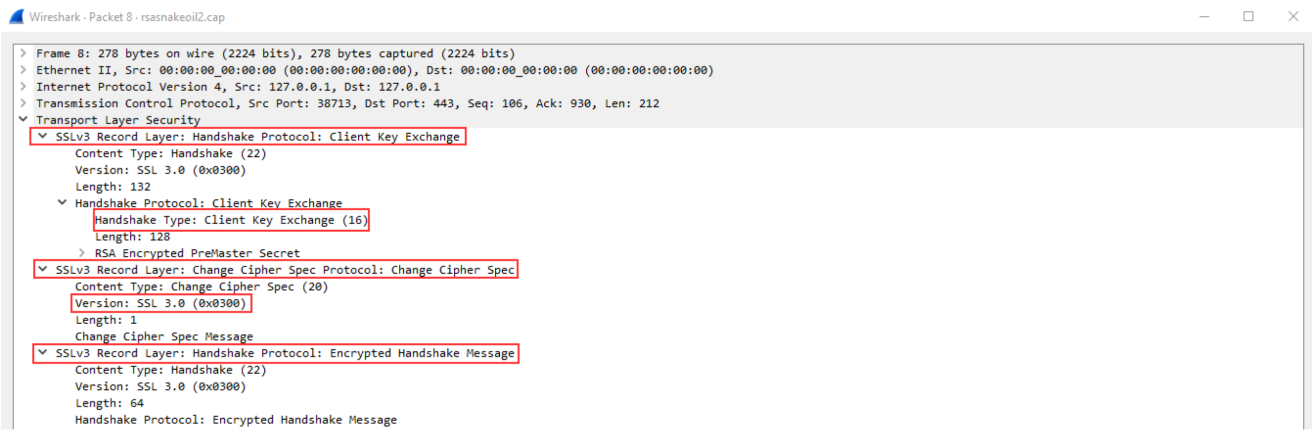
We can begin analyzing HTTPS traffic by looking at packets for the handshake between the client and the server. Below is a Client Hello packet showing the SSLv2 Record Layer, Handshake Type, and SSL Version.



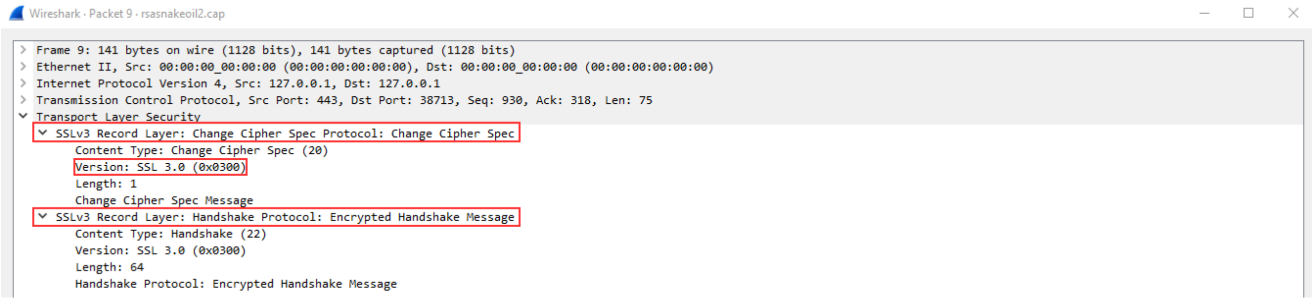
Below is the Server Hello packet sending similar information as the Client Hello packet however this time it includes session details and SSL certificate information



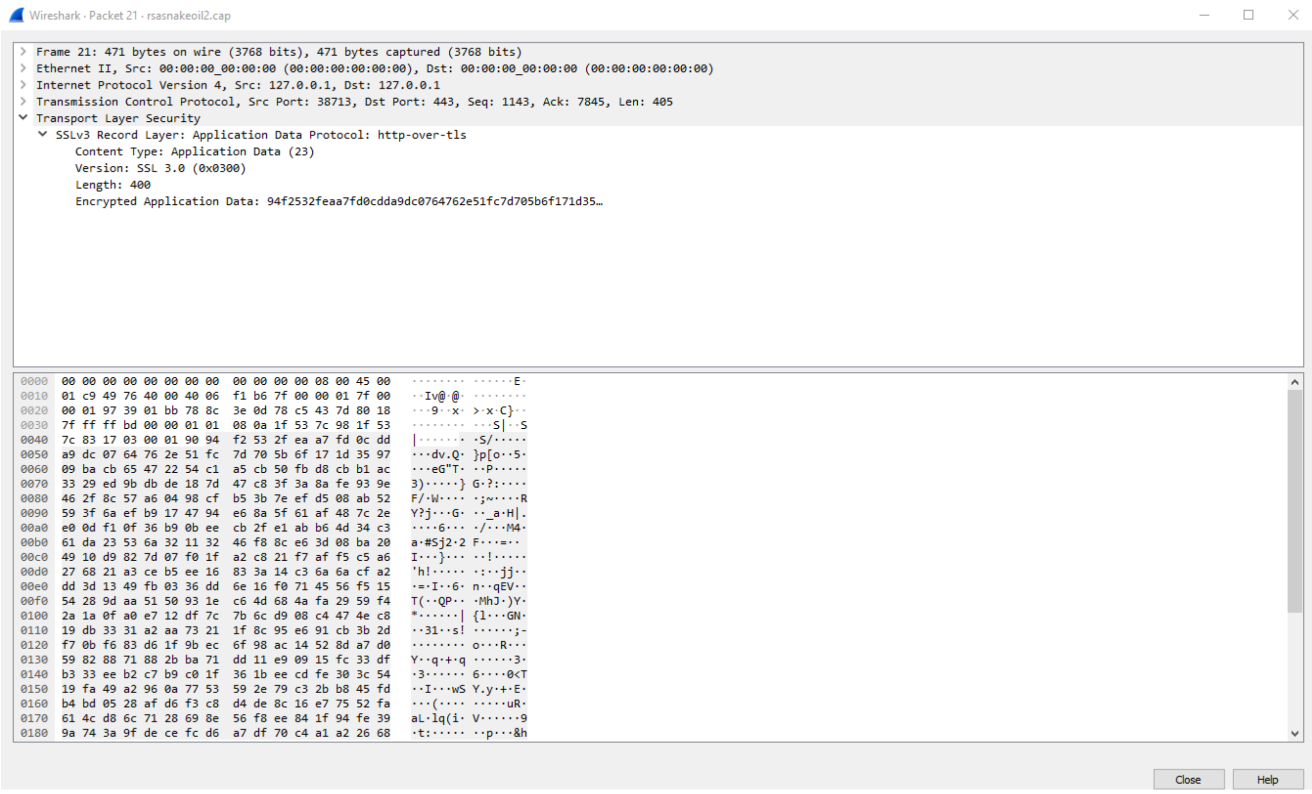
Below is the Client Key Exchange packet, this part of the handshake will determine the public key to use to encrypt further messages between the Client and Server.



In the next packet, the server will confirm the public key and create the secure tunnel, all traffic after this point will be encrypted based on the agreed-upon specifications listed above.



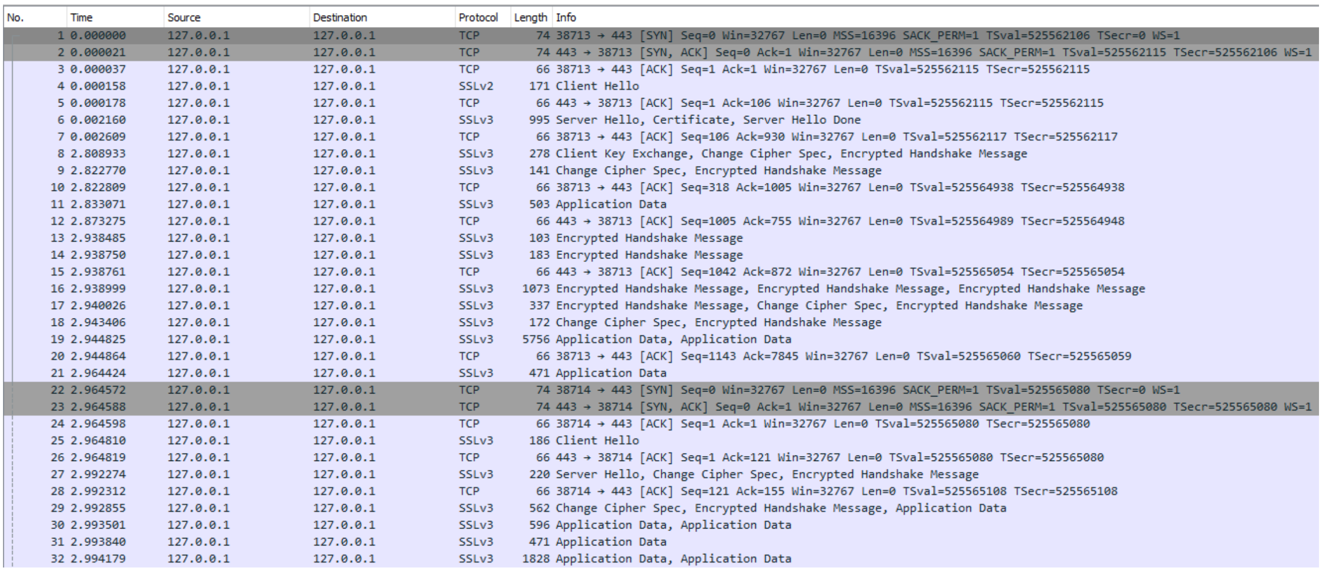
The traffic between the Client and the Server is now encrypted and you will need the secret key in order to decrypt the data stream being sent between the two hosts.



Practical HTTPS Packet Analysis

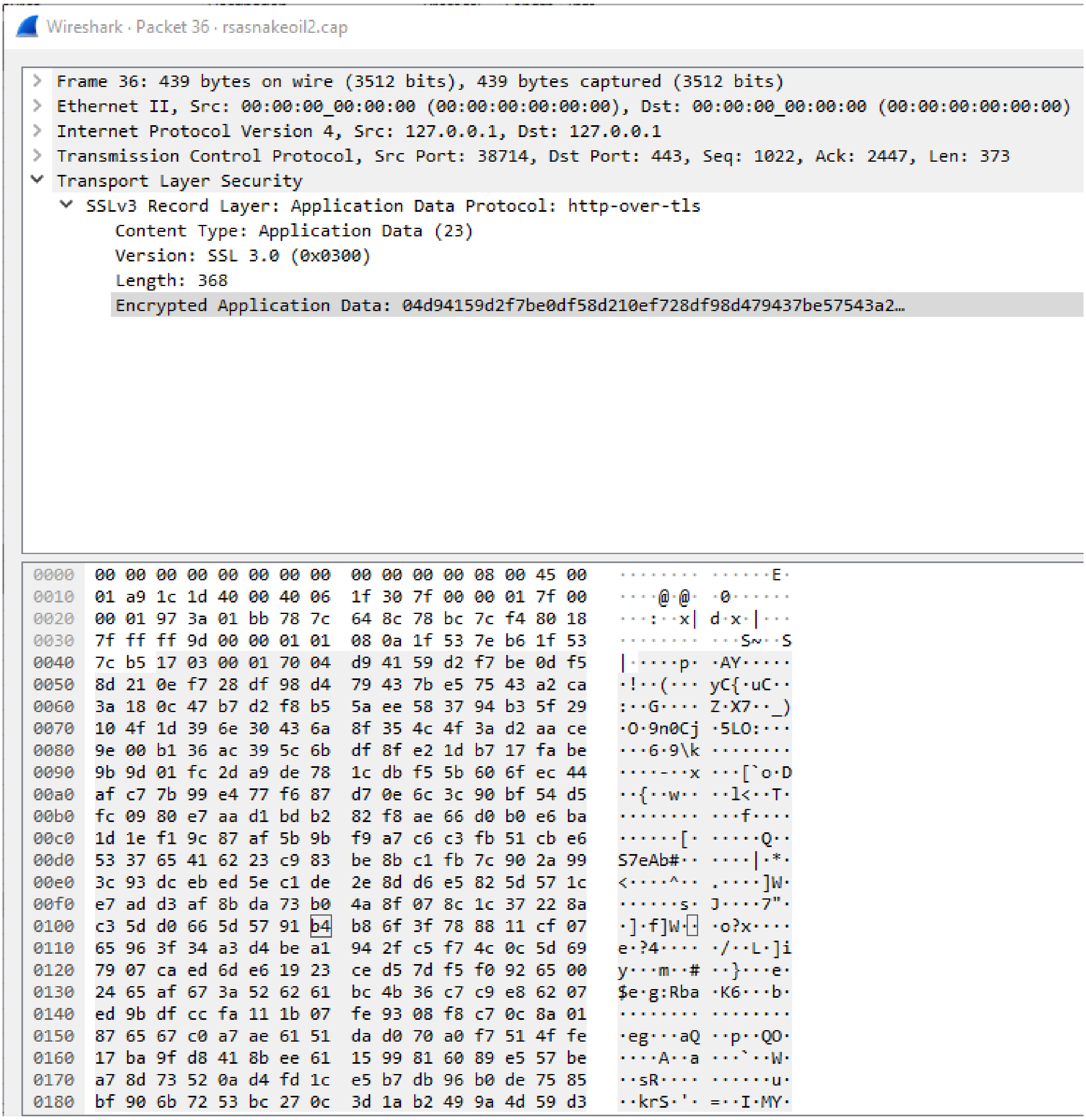
In order to practice and get hands-on with HTTPS packets, we can analyze the snakeoil2_070531 PCAP and decryption key set provided by Wireshark. Download the needed files from this task or directly from the Wireshark website.

We first need to load the PCAP into Wireshark. Navigate to File > Open and select the snakeoil2 PCAP.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	38713 → 443 [SYN] Seq=0 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525562106 TSecr=0 WS=1
2	0.000021	127.0.0.1	127.0.0.1	TCP	74	443 → 38713 [SYN, ACK] Seq=0 Ack=1 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525562115 TSecr=525562106 WS=1
3	0.000037	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1 Ack=1 Win=32767 Len=0 TSval=525562115 TSecr=525562115
4	0.000158	127.0.0.1	127.0.0.1	SSLv2	171	Client Hello
5	0.000178	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1 Ack=106 Win=32767 Len=0 TSval=525562115 TSecr=525562115
6	0.002160	127.0.0.1	127.0.0.1	SSLv3	995	Server Hello, Certificate, Server Hello Done
7	0.002609	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=106 Ack=930 Win=32767 Len=0 TSval=525562117 TSecr=525562117
8	2.808933	127.0.0.1	127.0.0.1	SSLv3	278	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9	2.822770	127.0.0.1	127.0.0.1	SSLv3	141	Change Cipher Spec, Encrypted Handshake Message
10	2.822809	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=318 Ack=1005 Win=32767 Len=0 TSval=525564938 TSecr=525564938
11	2.833071	127.0.0.1	127.0.0.1	SSLv3	503	Application Data
12	2.873275	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1005 Ack=755 Win=32767 Len=0 TSval=525564989 TSecr=525564948
13	2.938485	127.0.0.1	127.0.0.1	SSLv3	103	Encrypted Handshake Message
14	2.938750	127.0.0.1	127.0.0.1	SSLv3	183	Encrypted Handshake Message
15	2.938761	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1042 Ack=872 Win=32767 Len=0 TSval=525565054 TSecr=525565054
16	2.938999	127.0.0.1	127.0.0.1	SSLv3	1073	Encrypted Handshake Message, Encrypted Handshake Message, Encrypted Handshake Message
17	2.940026	127.0.0.1	127.0.0.1	SSLv3	337	Encrypted Handshake Message, Change Cipher Spec, Encrypted Handshake Message
18	2.943406	127.0.0.1	127.0.0.1	SSLv3	172	Change Cipher Spec, Encrypted Handshake Message
19	2.944825	127.0.0.1	127.0.0.1	SSLv3	5756	Application Data, Application Data
20	2.944864	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1143 Ack=7845 Win=32767 Len=0 TSval=525565060 TSecr=525565059
21	2.964424	127.0.0.1	127.0.0.1	SSLv3	471	Application Data
22	2.964572	127.0.0.1	127.0.0.1	TCP	74	38714 → 443 [SYN] Seq=0 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525565080 TSecr=0 WS=1
23	2.964588	127.0.0.1	127.0.0.1	TCP	74	443 → 38714 [SYN, ACK] Seq=0 Ack=1 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525565080 TSecr=525565080 WS=1
24	2.964598	127.0.0.1	127.0.0.1	TCP	66	38714 → 443 [ACK] Seq=1 Ack=1 Win=32767 Len=0 TSval=525565080 TSecr=525565080
25	2.964810	127.0.0.1	127.0.0.1	SSLv3	186	Client Hello
26	2.964819	127.0.0.1	127.0.0.1	TCP	66	443 → 38714 [ACK] Seq=1 Ack=121 Win=32767 Len=0 TSval=525565080 TSecr=525565080
27	2.992274	127.0.0.1	127.0.0.1	SSLv3	220	Server Hello, Change Cipher Spec, Encrypted Handshake Message
28	2.992312	127.0.0.1	127.0.0.1	TCP	66	38714 → 443 [ACK] Seq=121 Ack=155 Win=32767 Len=0 TSval=525565108 TSecr=525565108
29	2.992855	127.0.0.1	127.0.0.1	SSLv3	562	Change Cipher Spec, Encrypted Handshake Message, Application Data
30	2.993501	127.0.0.1	127.0.0.1	SSLv3	596	Application Data, Application Data
31	2.993840	127.0.0.1	127.0.0.1	SSLv3	471	Application Data
32	2.994179	127.0.0.1	127.0.0.1	SSLv3	1828	Application Data, Application Data

From looking at the above packet capture we can see that all of the requests are encrypted. Looking closer at the packets we can see the HTTPS handshake as well as the encrypted requests themselves. Let's take a closer look at one of the encrypted requests: Packet 11.



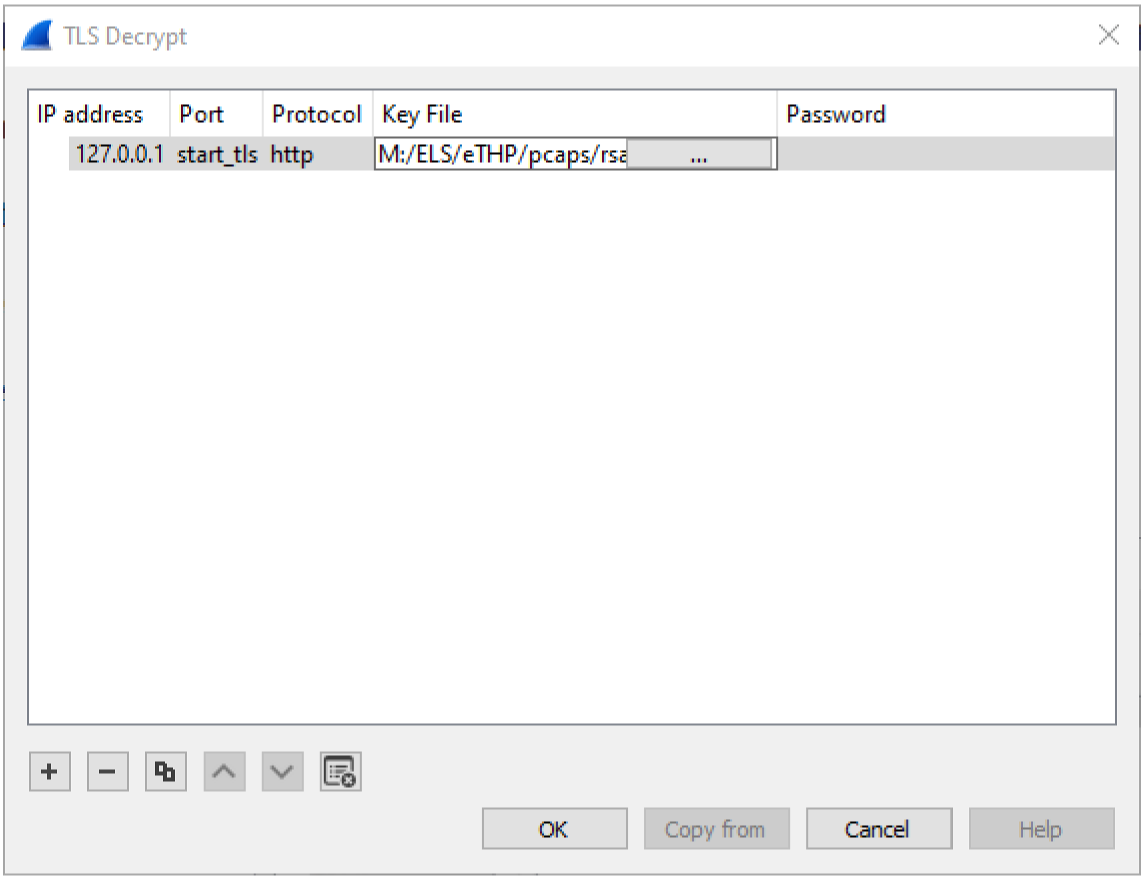
We can confirm from the packet details that the Application Data is encrypted. You can use an RSA key in Wireshark in order to view the data unencrypted. In order to load an RSA key navigate to Edit > Preferences > Protocols > TLS > [+] . If you are using an older version of Wireshark then this will be SSL instead of TLS. You will need to fill in the various sections on the menu with the following preferences:

IP Address: 127.0.0.1

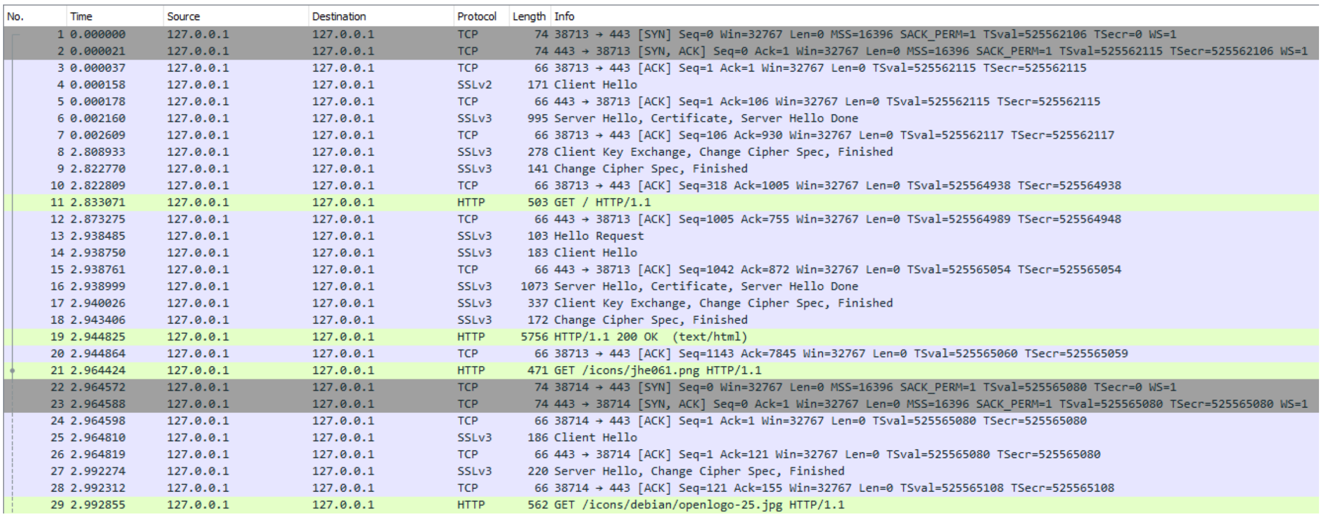
Port: start_tls

Protocol: http

Keyfile: RSA key location

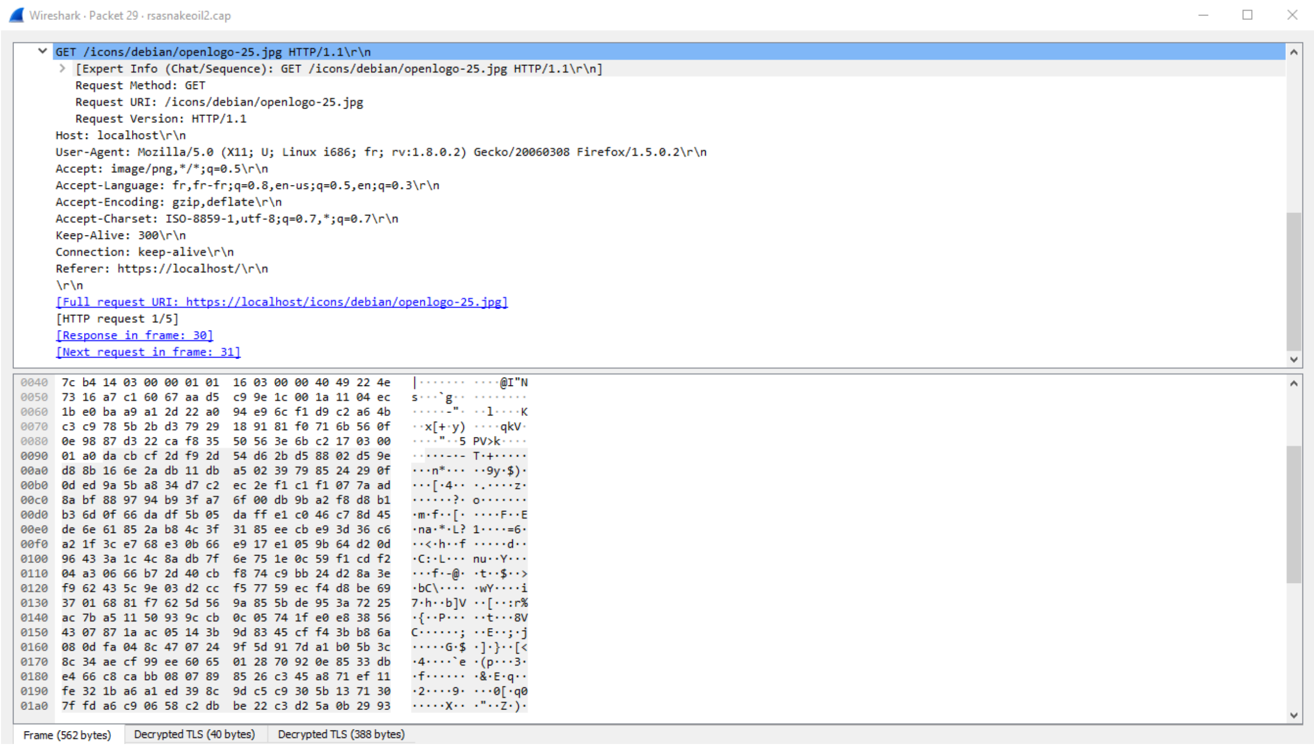


Now that we have an RSA key imported into Wireshark, if we go back to the packet capture we can see that the data stream is now unencrypted.



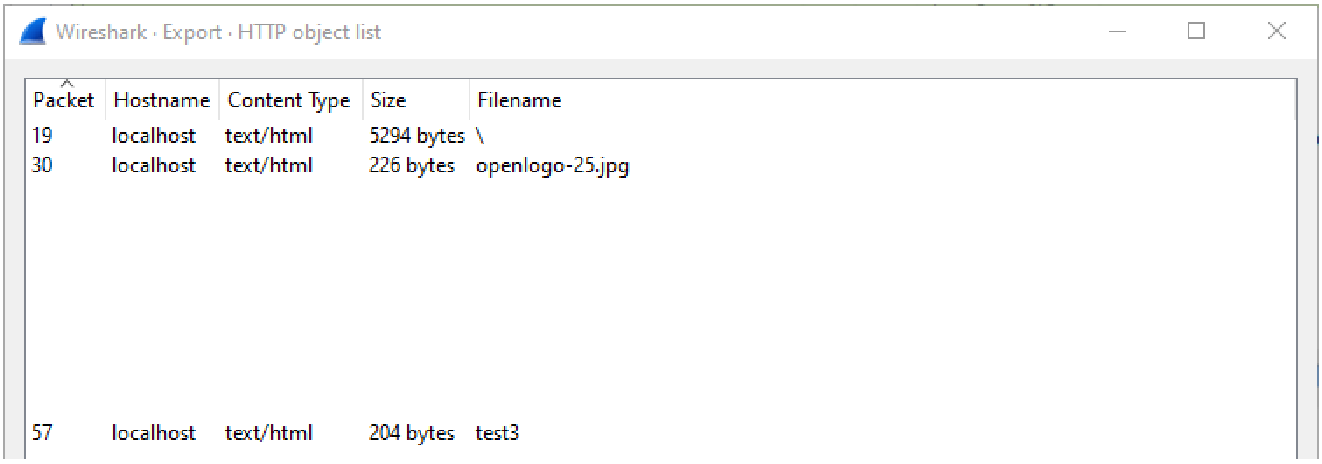
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	38713 → 443 [SYN] Seq=0 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525562106 TSecr=0 WS=1
2	0.000021	127.0.0.1	127.0.0.1	TCP	74	443 → 38713 [SYN, ACK] Seq=0 Ack=1 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525562115 TSecr=525562106 WS=1
3	0.000037	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1 Ack=1 Win=32767 Len=0 TSval=525562115 TSecr=525562115
4	0.000158	127.0.0.1	127.0.0.1	SSLv2	171	Client Hello
5	0.000178	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1 Ack=106 Win=32767 Len=0 TSval=525562115 TSecr=525562115
6	0.002160	127.0.0.1	127.0.0.1	SSLv3	995	Server Hello, Certificate, Server Hello Done
7	0.002609	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=106 Ack=930 Win=32767 Len=0 TSval=525562117 TSecr=525562117
8	2.808933	127.0.0.1	127.0.0.1	SSLv3	278	Client Key Exchange, Change Cipher Spec, Finished
9	2.822770	127.0.0.1	127.0.0.1	SSLv3	141	Change Cipher Spec, Finished
10	2.822809	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=318 Ack=1005 Win=32767 Len=0 TSval=525564938 TSecr=525564938
11	2.833071	127.0.0.1	127.0.0.1	HTTP	503	GET / HTTP/1.1
12	2.873275	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1005 Ack=755 Win=32767 Len=0 TSval=525564989 TSecr=525564948
13	2.938485	127.0.0.1	127.0.0.1	SSLv3	103	Hello Request
14	2.938750	127.0.0.1	127.0.0.1	SSLv3	183	Client Hello
15	2.938761	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1042 Ack=872 Win=32767 Len=0 TSval=525565054 TSecr=525565054
16	2.938999	127.0.0.1	127.0.0.1	SSLv3	1073	Server Hello, Certificate, Server Hello Done
17	2.940026	127.0.0.1	127.0.0.1	SSLv3	337	Client Key Exchange, Change Cipher Spec, Finished
18	2.943406	127.0.0.1	127.0.0.1	SSLv3	172	Change Cipher Spec, Finished
19	2.944825	127.0.0.1	127.0.0.1	HTTP	5756	HTTP/1.1 200 OK (text/html)
20	2.944864	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1143 Ack=7845 Win=32767 Len=0 TSval=525565060 TSecr=525565059
21	2.964424	127.0.0.1	127.0.0.1	HTTP	471	GET /icons/jhe061.png HTTP/1.1
22	2.964572	127.0.0.1	127.0.0.1	TCP	74	38714 → 443 [SYN] Seq=0 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525565080 TSecr=0 WS=1
23	2.964588	127.0.0.1	127.0.0.1	TCP	74	443 → 38714 [SYN, ACK] Seq=0 Ack=1 Win=32767 Len=0 MSS=16396 SACK_PERM=1 TSval=525565080 TSecr=525565080 WS=1
24	2.964598	127.0.0.1	127.0.0.1	TCP	66	38714 → 443 [ACK] Seq=1 Ack=1 Win=32767 Len=0 TSval=525565080 TSecr=525565080
25	2.964810	127.0.0.1	127.0.0.1	SSLv3	186	Client Hello
26	2.964819	127.0.0.1	127.0.0.1	TCP	66	443 → 38714 [ACK] Seq=1 Ack=121 Win=32767 Len=0 TSval=525565080 TSecr=525565080
27	2.992274	127.0.0.1	127.0.0.1	SSLv3	220	Server Hello, Change Cipher Spec, Finished
28	2.992312	127.0.0.1	127.0.0.1	TCP	66	38714 → 443 [ACK] Seq=121 Ack=155 Win=32767 Len=0 TSval=525565108 TSecr=525565108
29	2.992855	127.0.0.1	127.0.0.1	HTTP	562	GET /icons/debian/openlogo-25.jpg HTTP/1.1

We can now see the HTTP requests in unencrypted data streams. Looking further at one of the details of the packet we can see the unencrypted data stream closer.



Looking at the packet details we can see some very important information such as the request URI and the User-Agent which can be very useful in practical applications of Wireshark such as threat hunting and network administration.

We can now use other features in order to organize the data stream, like using the export HTTP object feature, to access this feature navigate to File > Export Objects > HTTP



Analyzing Exploit PCAPs

Zerologon PCAP Overview

We have gathered PCAP files from a recent Windows Active Directory Exploit called Zerologon or CVE-2020-1472. The scenario within the PCAP file contains a Windows Domain Controller with a private IP of 192.168.100.6 and an attacker with the private IP of 192.168.100.128. Let's walk through the steps of analyzing the PCAP and coming to a hypothesis of the events that happened.

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	54.193.240.194	192.168.100.128	OpenVPN	158	MessageType: P_DATA_V2
2	0.660801	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
3	1.662661	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
4	2.665708	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
5	3.031646	192.168.100.128	54.193.240.194	OpenVPN	158	MessageType: P_DATA_V2
6	3.665770	192.168.100.1	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
7	5.880142	54.193.240.194	192.168.100.128	OpenVPN	158	MessageType: P_DATA_V2
8	5.980996	192.168.100.128	192.168.100.6	TCP	74	60368 → 135 [SYN] Seq=658838935 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547159 TSecr=0 WS=128
9	5.981332	VMware_fc:eb:3a	Broadcast	ARP	42	Who has 192.168.100.128? Tell 192.168.100.6
10	5.981663	VMware_5f:4e:63	VMware_fc:eb:3a	ARP	60	192.168.100.128 is at 00:0c:29:5f:4e:63
11	5.981737	192.168.100.6	192.168.100.128	TCP	66	135 → 60368 [SYN, ACK] Seq=1736896598 Ack=658838936 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
12	5.982097	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658838936 Ack=1736896599 Win=64256 Len=0
13	5.982538	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: EPMv4 V3.0 (32bit NDR)
14	5.982638	192.168.100.6	192.168.100.128	DCERPC	114	Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4280 max_recv: 4280, 1 results: Acceptance
15	5.982917	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658839008 Ack=1736896659 Win=64256 Len=0
16	5.984650	192.168.100.128	192.168.100.6	EPM	210	Map request, RPC_NETLOGON, 32bit NDR
17	5.984933	192.168.100.6	192.168.100.128	EPM	206	Map response, RPC_NETLOGON, 32bit NDR
18	5.985196	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658839164 Ack=1736896811 Win=64128 Len=0
19	5.987270	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [FIN, ACK] Seq=658839164 Ack=1736896811 Win=64128 Len=0
20	5.987372	192.168.100.6	192.168.100.128	TCP	54	135 → 60368 [ACK] Seq=1736896811 Ack=658839165 Win=2102016 Len=0
21	5.987461	192.168.100.6	192.168.100.128	TCP	54	135 → 60368 [FIN, ACK] Seq=1736896811 Ack=658839165 Win=2102016 Len=0
22	5.987579	192.168.100.128	192.168.100.6	TCP	74	57936 → 49672 [SYN] Seq=2297288358 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547166 TSecr=0 WS=128
23	5.987660	192.168.100.6	192.168.100.128	TCP	66	49672 → 57936 [SYN, ACK] Seq=4257177707 Ack=2297288359 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
24	5.987791	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658839165 Ack=1736896812 Win=64128 Len=0
25	5.987980	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288359 Ack=4257177708 Win=64256 Len=0
26	5.988451	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: RPC_NETLOGON V1.0 (32bit NDR)
27	5.988584	192.168.100.6	192.168.100.128	DCERPC	114	Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4280 max_recv: 4280, 1 results: Acceptance
28	5.988875	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288431 Ack=4257177768 Win=64256 Len=0
29	5.990526	192.168.100.128	192.168.100.6	RPC_NE…	140	NetrServerReqChallenge request, DC01
30	5.990906	192.168.100.6	192.168.100.128	RPC_NE…	90	NetrServerReqChallenge response
31	5.991262	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288517 Ack=4257177804 Win=64256 Len=0
32	5.993544	192.168.100.128	192.168.100.6	RPC_NE…	174	NetrServerAuthenticate3 request
33	5.994367	192.168.100.6	192.168.100.128	RPC_NE…	98	NetrServerAuthenticate3 response, STATUS_ACCESS_DENIED

Identifying the Attacker

Immediately upon opening the PCAP file we see some things that may be out of the ordinary. First, we see some normal traffic from OpenVPN, ARP, etc. We then start to identify what would be known as unknown protocols in this case DCERPC and EPM.

Looking at the packets we see that 192.168.100.128 is sending all of the requests, so we can assume that the device is the attacker. We can continue looking at packets coming from this IP to narrow down our hunt.

Zerologon POC Connection Analysis

ip.src == 192.168.100.128

No.	Time	Source	Destination	Protocol	Length	Info
5	3.031646	192.168.100.128	54.193.240.194	OpenVPN	158	MessageType: P_DATA_V2
8	5.980996	192.168.100.128	192.168.100.6	TCP	74	60368 → 135 [SYN] Seq=658838935 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547159 TSecr=0 WS=128
12	5.982097	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658838936 Ack=1736896599 Win=64256 Len=0
13	5.982538	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: EPMv4 V3.0 (32bit NDR)
15	5.982917	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658839008 Ack=1736896659 Win=64256 Len=0
16	5.984650	192.168.100.128	192.168.100.6	EPM	210	Map request, RPC_NETLOGON, 32bit NDR
18	5.985196	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658839164 Ack=1736896811 Win=64128 Len=0
19	5.987270	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [FIN, ACK] Seq=658839164 Ack=1736896811 Win=64128 Len=0
22	5.987579	192.168.100.128	192.168.100.6	TCP	74	57936 → 49672 [SYN] Seq=2297288358 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547166 TSecr=0 WS=128
24	5.987791	192.168.100.128	192.168.100.6	TCP	60	60368 → 135 [ACK] Seq=658839165 Ack=1736896812 Win=64128 Len=0
25	5.987980	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288359 Ack=4257177708 Win=64256 Len=0
26	5.988451	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: RPC_NETLOGON V1.0 (32bit NDR)
28	5.988875	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288431 Ack=4257177768 Win=64256 Len=0
29	5.990526	192.168.100.128	192.168.100.6	RPC_NE…	140	NetrServerReqChallenge request, DC01
31	5.991262	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288517 Ack=4257177804 Win=64256 Len=0
32	5.993544	192.168.100.128	192.168.100.6	RPC_NE…	174	NetrServerAuthenticate3 request
34	5.994738	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288637 Ack=4257177848 Win=64256 Len=0
35	5.995998	192.168.100.128	192.168.100.6	TCP	74	60372 → 135 [SYN] Seq=1240177321 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547174 TSecr=0 WS=128
37	5.996381	192.168.100.128	192.168.100.6	TCP	60	60372 → 135 [ACK] Seq=1240177322 Ack=2538286179 Win=64256 Len=0
38	5.996840	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: EPMv4 V3.0 (32bit NDR)
40	5.997177	192.168.100.128	192.168.100.6	TCP	60	60372 → 135 [ACK] Seq=1240177394 Ack=2538286239 Win=64256 Len=0
41	5.999869	192.168.100.128	192.168.100.6	EPM	210	Map request, RPC_NETLOGON, 32bit NDR
43	6.000525	192.168.100.128	192.168.100.6	TCP	60	60372 → 135 [ACK] Seq=1240177550 Ack=2538286391 Win=64128 Len=0
44	6.002161	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [FIN, ACK] Seq=2297288637 Ack=4257177848 Win=64256 Len=0
47	6.002618	192.168.100.128	192.168.100.6	TCP	60	57936 → 49672 [ACK] Seq=2297288638 Ack=4257177849 Win=64256 Len=0
48	6.003661	192.168.100.128	192.168.100.6	TCP	60	60372 → 135 [FIN, ACK] Seq=1240177550 Ack=2538286391 Win=64128 Len=0
50	6.004262	192.168.100.128	192.168.100.6	TCP	74	57940 → 49672 [SYN] Seq=4278901033 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051547182 TSecr=0 WS=128
53	6.004686	192.168.100.128	192.168.100.6	TCP	60	60372 → 135 [ACK] Seq=1240177551 Ack=2538286392 Win=64128 Len=0
54	6.004927	192.168.100.128	192.168.100.6	TCP	60	57940 → 49672 [ACK] Seq=4278901034 Ack=3697163111 Win=64256 Len=0
55	6.005855	192.168.100.128	192.168.100.6	DCERPC	126	Bind: call_id: 1, Fragment: Single, 1 context items: RPC_NETLOGON V1.0 (32bit NDR)
57	6.006420	192.168.100.128	192.168.100.6	TCP	60	57940 → 49672 [ACK] Seq=4278901106 Ack=3697163171 Win=64256 Len=0
58	6.008464	192.168.100.128	192.168.100.6	RPC_NE…	140	NetrServerReqChallenge request, DC01
60	6.009096	192.168.100.128	192.168.100.6	TCP	60	57940 → 49672 [ACK] Seq=4278901192 Ack=3697163207 Win=64256 Len=0
61	6.011129	192.168.100.128	192.168.100.6	RPC_NE…	174	NetrServerAuthenticate3 request

We can set a filter for the src of the IP that we believe to be suspicious. When analyzing PCAPS we need to be aware of IOCs or Indicators of Compromise particular exploits may have with them. This is known as Threat Intelligence, which is out of the scope of this room; I recommend that after completing this room if you're interested more then do your own research on the topic. In this case, if we had background knowledge of the Zerologon exploit, we would know that the exploit uses multiple RPC connections, and DCERPC requests to change the machine account password, which could be verified with the PCAP.

Secretsdump SMB Analysis

Looking further at the PCAP we can see SMB2/3 traffic and DRSUAPI traffic, again with prior knowledge of the attack we know that it uses secretsdump to dump hashes. Secretsdump abuses SMB2/3 and DRSUAPI to do this, so we can assume that this traffic is secretsdump.

1093	25.617789	192.168.100.128	192.168.100.6	SMB2	270	Encrypted SMB3
1095	25.621367	192.168.100.128	192.168.100.6	SMB2	223	Encrypted SMB3
1097	25.625251	192.168.100.128	192.168.100.6	DRSUAPI	274	DsCrackNames request
1099	25.629925	192.168.100.128	192.168.100.6	DRSUAPI	394	DsGetNCChanges request
1102	25.631491	192.168.100.128	192.168.100.6	TCP	60	47770 → 49668 [ACK] Seq=698962605 Ack=2922578794 Win=61568 Len=0
1103	25.652358	192.168.100.128	192.168.100.6	SMB2	270	Encrypted SMB3
1105	25.655229	192.168.100.128	192.168.100.6	SMB2	223	Encrypted SMB3
1107	25.658381	192.168.100.128	192.168.100.6	DRSUAPI	274	DsCrackNames request
1109	25.662692	192.168.100.128	192.168.100.6	DRSUAPI	394	DsGetNCChanges request
1112	25.663917	192.168.100.128	192.168.100.6	TCP	60	47770 → 49668 [ACK] Seq=698963165 Ack=2922583786 Win=61568 Len=0
1113	25.679773	192.168.100.128	192.168.100.6	SMB2	270	Encrypted SMB3
1115	25.683946	192.168.100.128	192.168.100.6	SMB2	223	Encrypted SMB3
1117	25.688642	192.168.100.128	192.168.100.6	DRSUAPI	274	DsCrackNames request
1119	25.692958	192.168.100.128	192.168.100.6	DRSUAPI	394	DsGetNCChanges request
1122	25.694400	192.168.100.128	192.168.100.6	TCP	60	47770 → 49668 [ACK] Seq=698963725 Ack=2922588282 Win=62592 Len=0
1123	25.711316	192.168.100.128	192.168.100.6	TCP	74	47012 → 445 [SYN] Seq=2291350219 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3051566889 TSecr=0 WS=128
1125	25.711789	192.168.100.128	192.168.100.6	TCP	60	47012 → 445 [ACK] Seq=2291350220 Ack=2158419710 Win=64256 Len=0
1126	25.712407	192.168.100.128	192.168.100.6	SMB	127	Negotiate Protocol Request
1128	25.713145	192.168.100.128	192.168.100.6	TCP	60	47012 → 445 [ACK] Seq=2291350293 Ack=2158419962 Win=64128 Len=0
1129	25.714684	192.168.100.128	192.168.100.6	SMB2	164	Negotiate Protocol Request
1131	25.715693	192.168.100.128	192.168.100.6	TCP	60	47012 → 445 [ACK] Seq=2291350403 Ack=2158420214 Win=64128 Len=0
1132	25.718152	192.168.100.128	192.168.100.6	SMB2	212	Session Setup Request, NTLMSSP_NEGOTIATE
1134	25.718666	192.168.100.128	192.168.100.6	TCP	60	47012 → 445 [ACK] Seq=2291350561 Ack=2158420547 Win=64128 Len=0
1135	25.720404	192.168.100.128	192.168.100.6	SMB2	500	Session Setup Request, NTLMSSP_AUTH, User: \DC01$
1137	25.722211	192.168.100.128	192.168.100.6	TCP	60	47012 → 445 [ACK] Seq=2291351007 Ack=2158420632 Win=64128 Len=0
1138	25.724287	192.168.100.128	192.168.100.6	SMB2	222	Encrypted SMB3
1140	25.726571	192.168.100.128	192.168.100.6	SMB2	242	Encrypted SMB3
1142	25.729843	192.168.100.128	192.168.100.6	SMB2	294	Encrypted SMB3

Each exploit and attack will come with its unique artifacts, in this case, it is clear what happened and the order of events that occurred. Once we have identified the attacker we would need to move on to other steps to identify and isolate as well as report the incident if we were on a Threat Hunting or DFIR team.