HideZeroOne
INE – Cyber Sec
www.hide01.ir

Incident Handling & Response Professional

v1

Intrusion Detection By Analyzing Traffic (Part 1)

Section 02 | Module 01

OUTLINE

Table of Contents

Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ Networking and traffic analysis concepts
- ✓ How the attacks at the IEEE 802.x Link and IP layers work
- ✓ How to detect attacks in the IEEE 802.x Link and IP layers

1.1

Networking Concepts & Analysis

1.1 Networking Concepts & Analysis

As already mentioned in the previous section, part of your incident handling activities will be analyzing captured or live traffic.

Traffic analysis activities can be performed in multiple phases of the incident handling process.

1.1 Networking Concepts & Analysis

For example, there is no doubt you will perform traffic analysis activities during the Detection & Analysis phase of the incident handling process.

But, you will also have to perform the same during the Containment, Eradication & Recovery phase; specifically, during Long-term Containment and Recovery.

OUTLINE

1.1 Networking Concepts & Analysis

Let's first cover some networking concepts before we proceed to the more practical part of this module, where we will see how to detect intrusions or spot suspicious behavior on the wire, by analyzing network traffic.

1.1.1 Communication Models

An important IT concept is *communication model*.

Any network traffic's format, order and disassembly are based on communication models. Frames, packets, headers and data emulate the layers of network communication models.

1.1.1 Communication Models

Back in the day, a communication standard was established on how hosts would communicate within a network; this standard is the OSI (Open Systems Interconnection) model.

The OSI model consists of seven (7) layers. Each layer represents a different function that networked hosts will perform during communication.

1.1.1 Communication Models

Another model, the TCP/IP one, eventually gained more traction; this model consists of four (4) layers. That being said, the OSI and the TCP/IP models are quite similar, despite their representation differences.

The next slide contains a graphic representation of how the two models resemble each other conceptually.

1.1.1 Communication Models

For the context of this course, the terms **Packet** and **Datagram** are synonymous.

DATA	OSI MODEL	TCP MODEL	
Data	Application Network Process to Application	Application	HTTP, SMTP, DNS
Data	Presentation Data Representation and Encryption		
Data	Session Inter host Communication		
Segment	Transport End to End connection and reliability	Transport	TCP, UDP
Packet	Network Best path determination and IP (Logical) Addressing	Internet	IP
Frame	Data Link MAC and LLC (Physical Addressing)	Network Access	IEEE 802.x
Bits	Physical Media, Signal and Binary Transmission		

OUTLINE

1.1.1 Communication Models

Under the hood, during host communication, layers communicate with upper or lower layers, so that all communication pieces are gathered.

1.1.2 TCP/IP

Let's focus our attention on how the TCP/IP model works.

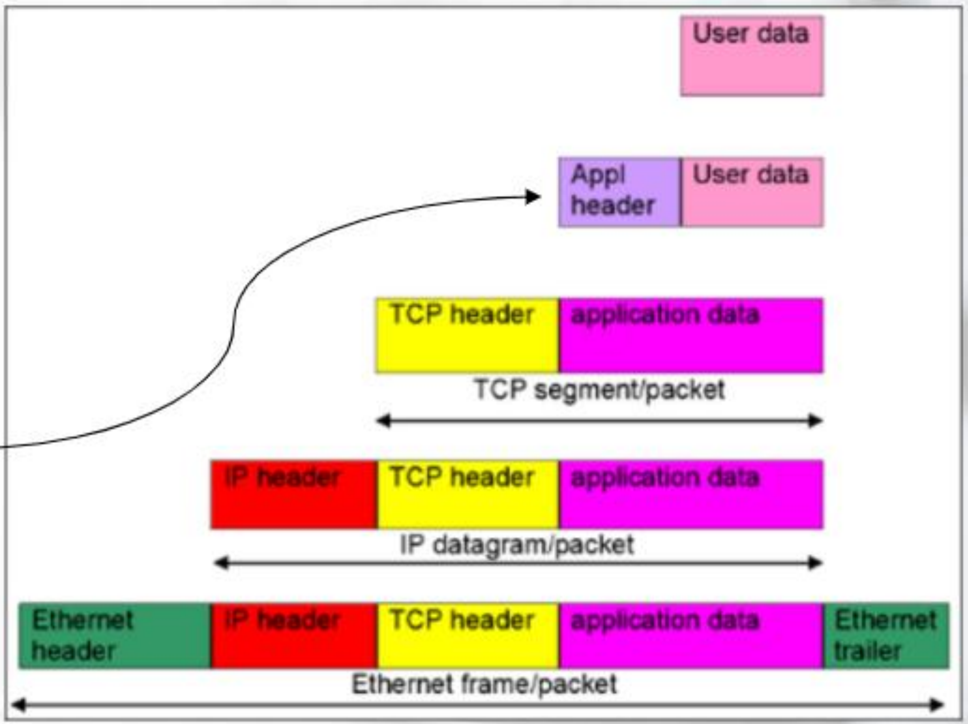
As previously mentioned, all communication pieces should be gathered from all layers before a message is sent.

More specifically, whenever data is being prepared by a host for transmission, each of the lower layers of the TCP/IP stack add something; this is known as *encapsulation of layers*.

1.1.2 TCP/IP

Encapsulation

Application Payload
At the Application layer, the respective payload or data is supplied. The payload or data will appear after all the encapsulating headers.

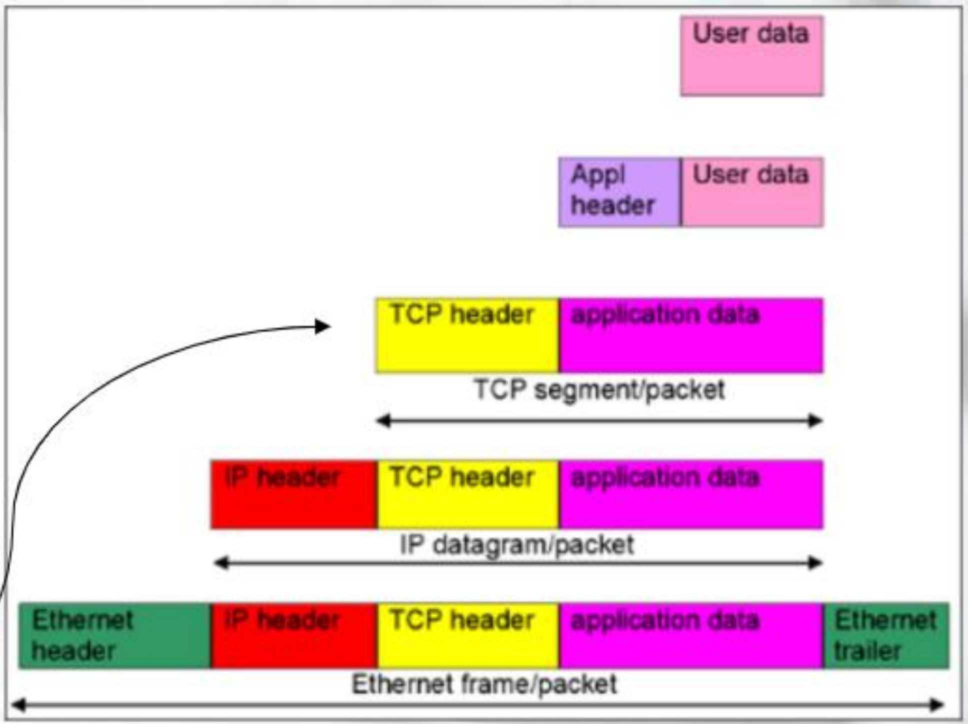


1.1.2 TCP/IP

Encapsulation

TCP Header
The Application layer passes the payload down to the Transport layer (in this case TCP). The Transport layer adds a TCP header to the application payload.

This header includes crucial transmission information such as source and destination ports as well as information that make sure the TCP segment arrives as expected.



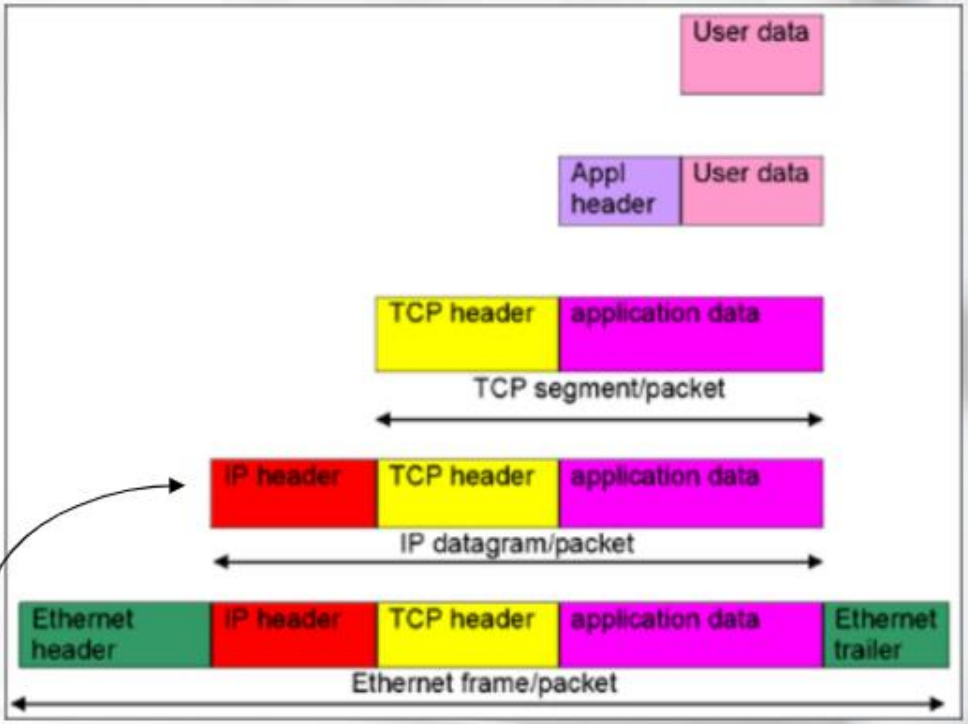
OUTLINE

1.1.2 TCP/IP

Encapsulation

IP Packet Header
The TCP header and the application payload are now being pushed to the Internet layer. As you can imagine, the Internet layer adds yet another header known as the IP header. This header includes information that makes sure the packet is delivered to the correct destination IP.

At this point, the TCP header can be seen as IP data. This is because, at this point, the IP or Networking layer is not interested in knowing information such as destination ports, etc.



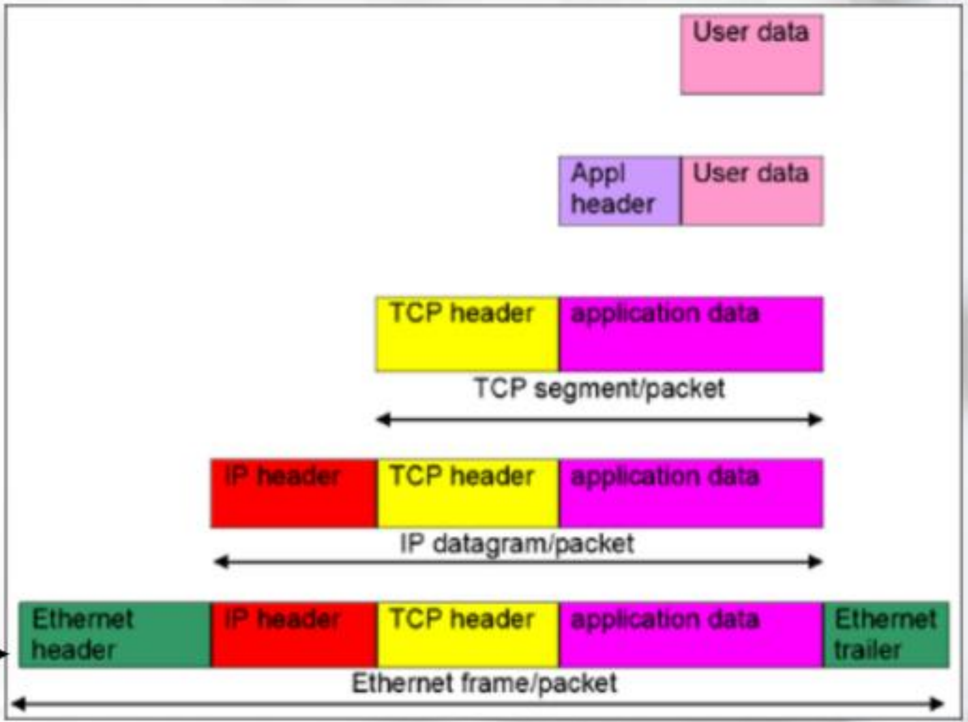
OUTLINE

1.1.2 TCP/IP

Encapsulation

Frame header
The IP and TCP headers, as well as the application payload (aka the IP packet), are now passed to the Network Access layer.

At this point, the Network Access layer adds a header to the IP packet known as the Frame header, containing information such as source and destination MAC addresses, etc. The IP packet can be seen as Frame data.



OUTLINE

1.1.2 TCP/IP

At the destination host, encapsulation will have to be reversed.

Each received frame will have to be stripped of its headers so that each resulting message is passed to the appropriate layer; this process is known as de-encapsulation.

1.1.2 TCP/IP

De-encapsulation

For example, the Network Access layer will receive the frame, analyze the data, strip the frame header off and pass the IP header and accompanying data to the Internet layer.

Each layer deals only with the data that are meant to be handled by it. Most of the time, a protocol identifier exists which can be found in the previous layer.

1.1.2 TCP/IP

De-encapsulation

It should be noted, that some layer headers are of fixed size, while others are of variable length size and are accompanied by length details (this is useful to discriminate optional data or know where a layer ends).

Based on this knowledge, each current layer knows where to start and stop processing data and subsequently what it should pass to the upper layer.

OUTLINE

1.1.2 TCP/IP

De-encapsulation

To conclude the topic of how each layer knows what to process, option fields also exist, residing in IP and TCP headers, that are used to identify where current layer options start and end.

Let's briefly see a frame de-encapsulation example.

1.1.2 TCP/IP

De-encapsulation

Let's suppose a network access card supports the Ethernet (802.3) link layer protocol.

This means that the card understands the format and all the fields in the frame header.

1.1.2 TCP/IP

OUTLINE

De-encapsulation

Layer		
Application	4.	Unfortunately, the Transport layer knows nothing about the protocol that follows or the application's data/protocol format. It only knows the destination port. We may be dealing with HTTP, but we are not sure.
Transport	3.	The IPv4 header, in turn, includes an indicator about the transport protocol that follows; this is the *protocol* field. Let's suppose the protocol is TCP. The data following the TCP header is passed to the Transport protocol.
Internet	2.	The Ethernet layer will pass all data following the Ethernet header to the IPv4-handling part of the IP layer.
Network Access	1.	For data to be passed properly to the IP layer, an indicator should exist that reveals what protocol follows the Ethernet header (could be IPv4 or IPv6); this is where the *type* field of the Ethernet header jumps in. Let's suppose the protocol is IPv4.

1.1.2 TCP/IP

De-encapsulation

Same example, but a little deeper…

Application	4.	The amount of data to be passed to the Application layer can be derived as follows. **Total IP datagram length - IP header length - TCP header length = Length of data to be passed**
Transport	3.	The Transport layer header could be of fixed length (UDP or ICMP) or of variable length (TCP). For example, we can find a header length value about the TCP header, so that the position of the data that follow the TCP header can be identified and passed to the Application layer.
Internet	2.	The standard IPv4 header has a length of 20 bytes; however, there are cases when IP option data exist. Those data can result in the IPv4 header expanding up to 60 bytes (variable length). The IP header contains a length-related field, so that data following the IP header are passed to the Transport layer. The IP header also has a field indicating the whole IP datagram length.
Network Access	1.	The Ethernet (802.3) header has a fixed length of 14 bytes; this means that the data that will be passed to the IP layer begin 14 bytes after the Ethernet header.

Encapsulation

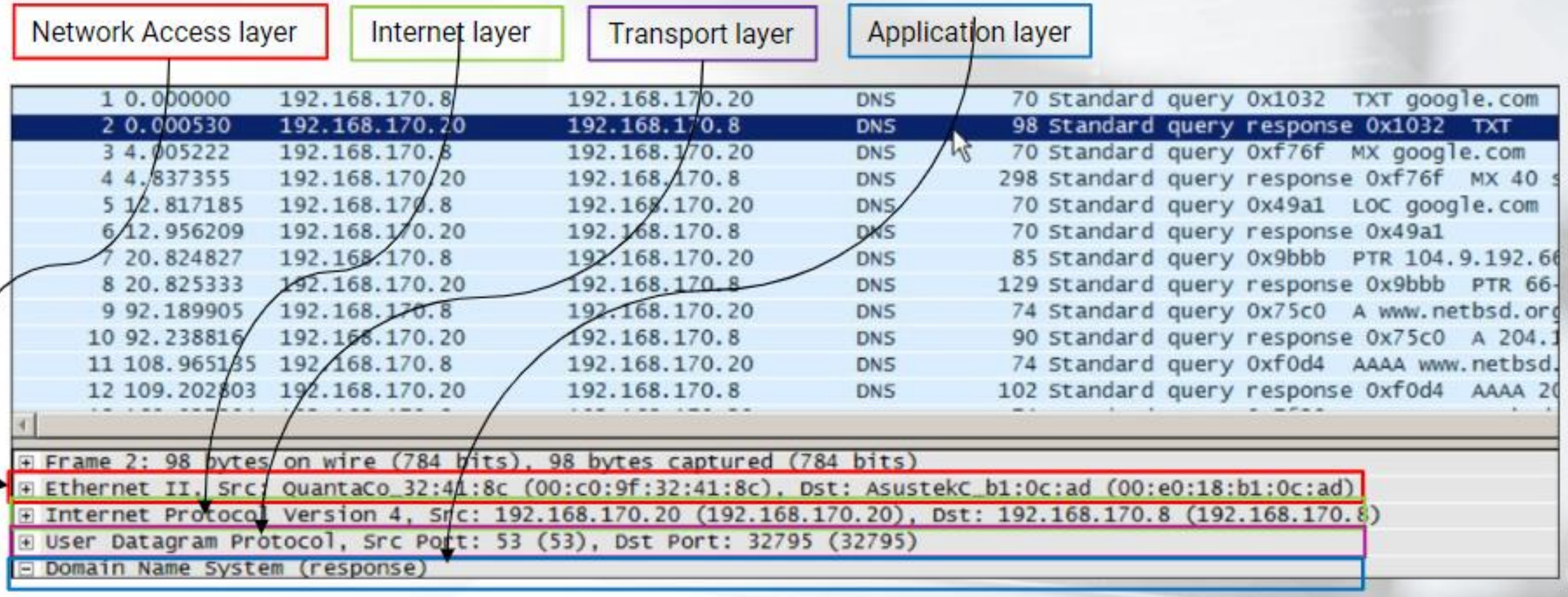
Traffic analysis tools, such as Wireshark, display data encapsulation. Just be careful in Wireshark's case, because the layers are displayed in reverse order (the lower layers are displayed first).

For example, let's see how a DNS response looks like in Wireshark, accompanied by encapsulation details.

https://www.wireshark.org/

1.1.2 TCP/IP

Encapsulation



OUTLINE
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP
1.1.2 TCP/IP

1.1.2 TCP/IP

Encapsulation

Network Access layer: In this line, Wireshark displays the source and destination MAC addresses.

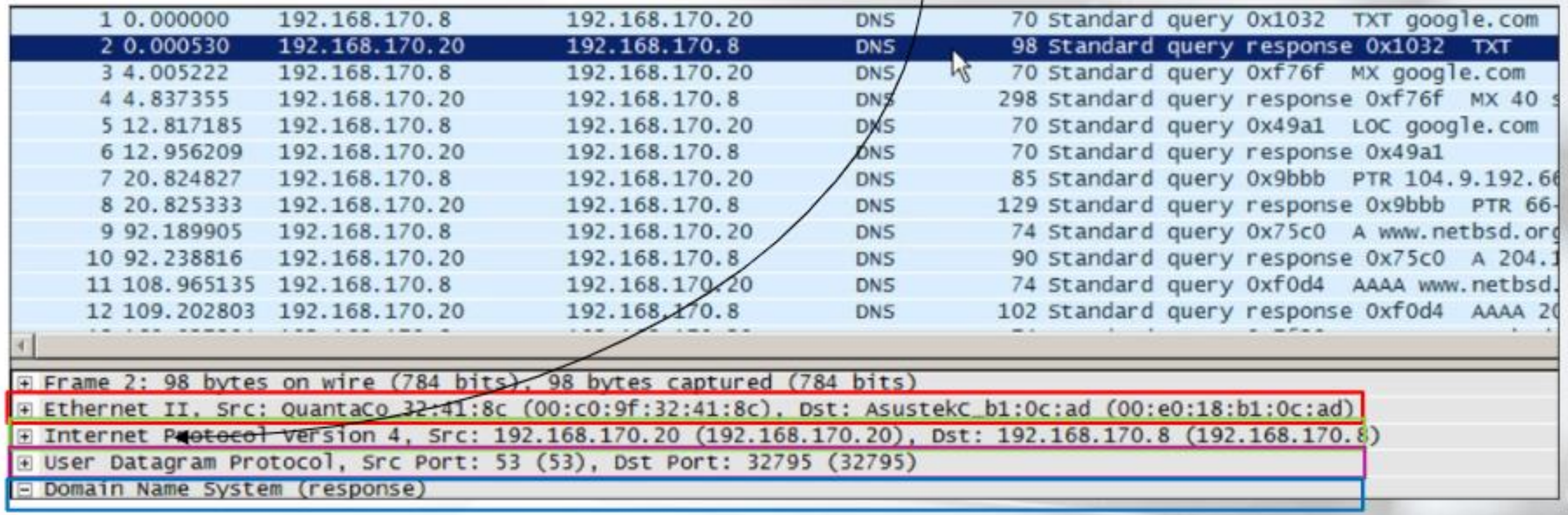
1	0.000000	192.168.170.8	192.168.170.20	DNS	70	Standard query 0x1032 TXT google.com
2	0.000530	192.168.170.20	192.168.170.8	DNS	98	Standard query response 0x1032 TXT
3	4.005222	192.168.170.8	192.168.170.20	DNS	70	Standard query 0xf76f MX google.com
4	4.837355	192.168.170.20	192.168.170.8	DNS	298	Standard query response 0xf76f MX 40 s
5	12.817185	192.168.170.8	192.168.170.20	DNS	70	Standard query 0x49a1 LOC google.com
6	12.956209	192.168.170.20	192.168.170.8	DNS	70	Standard query response 0x49a1
7	20.824827	192.168.170.8	192.168.170.20	DNS	85	Standard query 0x9bbb PTR 104.9.192.66
8	20.825333	192.168.170.20	192.168.170.8	DNS	129	Standard query response 0x9bbb PTR 66-
9	92.189905	192.168.170.8	192.168.170.20	DNS	74	Standard query 0x75c0 A www.netbsd.org
10	92.238816	192.168.170.20	192.168.170.8	DNS	90	Standard query response 0x75c0 A 204.1
11	108.965135	192.168.170.8	192.168.170.20	DNS	74	Standard query 0xf0d4 AAAA www.netbsd.
12	109.202803	192.168.170.20	192.168.170.8	DNS	102	Standard query response 0xf0d4 AAAA 20

```
⊞ Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
⊞ Ethernet II, Src: QuantaCo_32:41:8c (00:c0:9f:32:41:8c), Dst: AsustekC_b1:0c:ad (00:e0:18:b1:0c:ad)
⊞ Internet Protocol Version 4, Src: 192.168.170.20 (192.168.170.20), Dst: 192.168.170.8 (192.168.170.8)
⊞ User Datagram Protocol, Src Port: 53 (53), Dst Port: 32795 (32795)
⊟ Domain Name System (response)
```

1.1.2 TCP/IP

Encapsulation

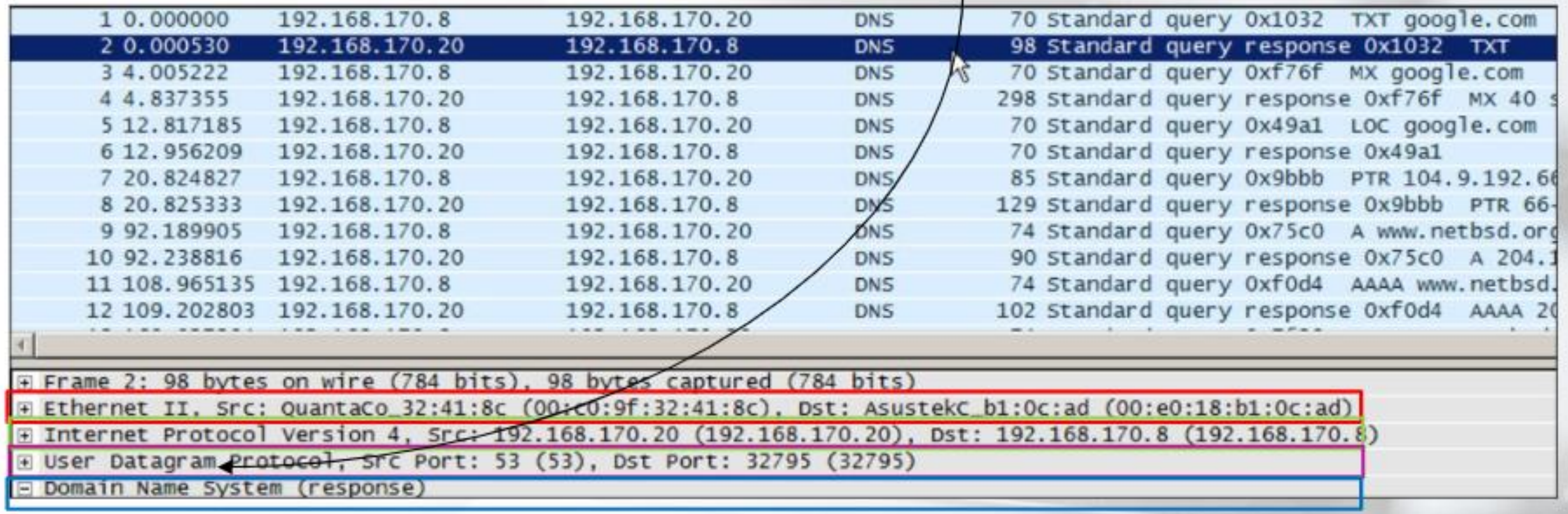
Internet layer: In this line, Wireshark displays the IP layer that comes right after the Frame header.



1.1.2 TCP/IP

Encapsulation

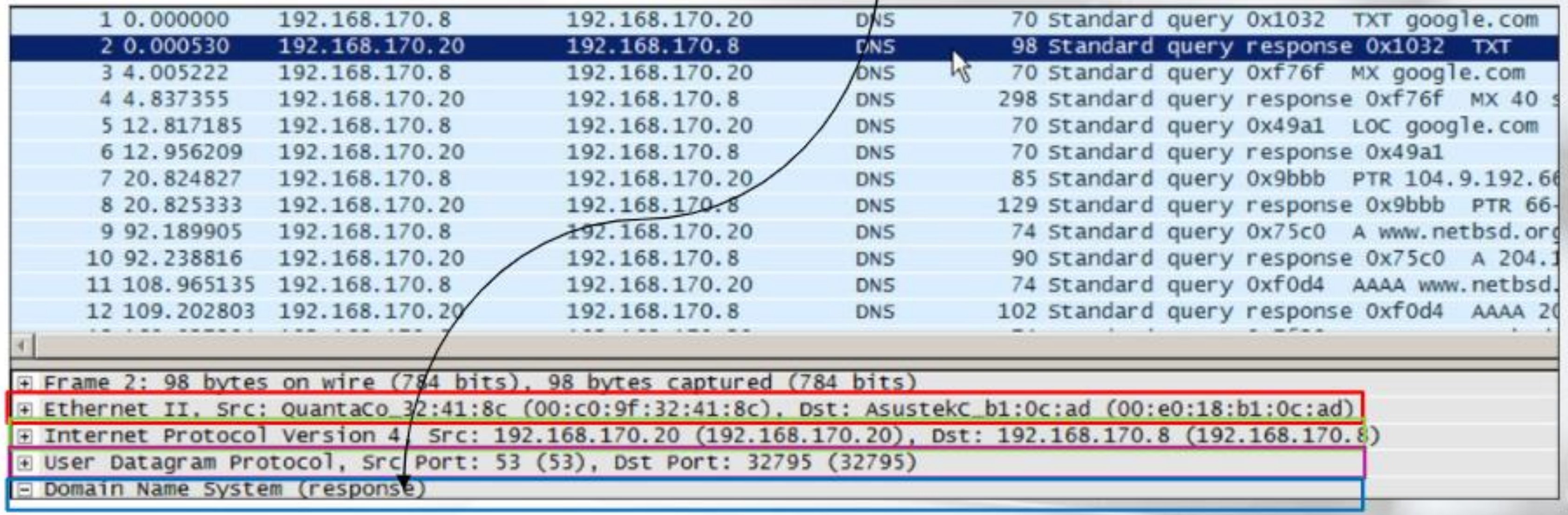
Transport layer: In this case, UDP. It follows the IP header (recall the TCP/IP stack). Source and destination ports are included in this summary line.



OUTLINE

1.1.2 TCP/IP

Application layer: This summary line, displays a description of the application payload, a DNS response.



1.1.2 TCP/IP

Encapsulation

We can have a detailed look at each encapsulated layer by expanding the respective fields.

```
⊞ Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
⊞ Ethernet II, Src: QuantaCo_32:41:8c (00:c0:9f:32:41:8c), Dst: AsustekC_b1:0c:ad (00:e0:18:b1:0c:ad)
⊞ Internet Protocol Version 4, Src: 192.168.170.20 (192.168.170.20), Dst: 192.168.170.8 (192.168.170.8)
⊞ User Datagram Protocol, Src Port: 53 (53), Dst Port: 32795 (32795)
⊟ Domain Name System (response)
    [Request In: 1]
    [Time: 0.000530000 seconds]
    Transaction ID: 0x1032
  ⊞ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊞ google.com: type TXT, class IN
  ⊞ Answers
```

1.1.2 TCP/IP

Encapsulation

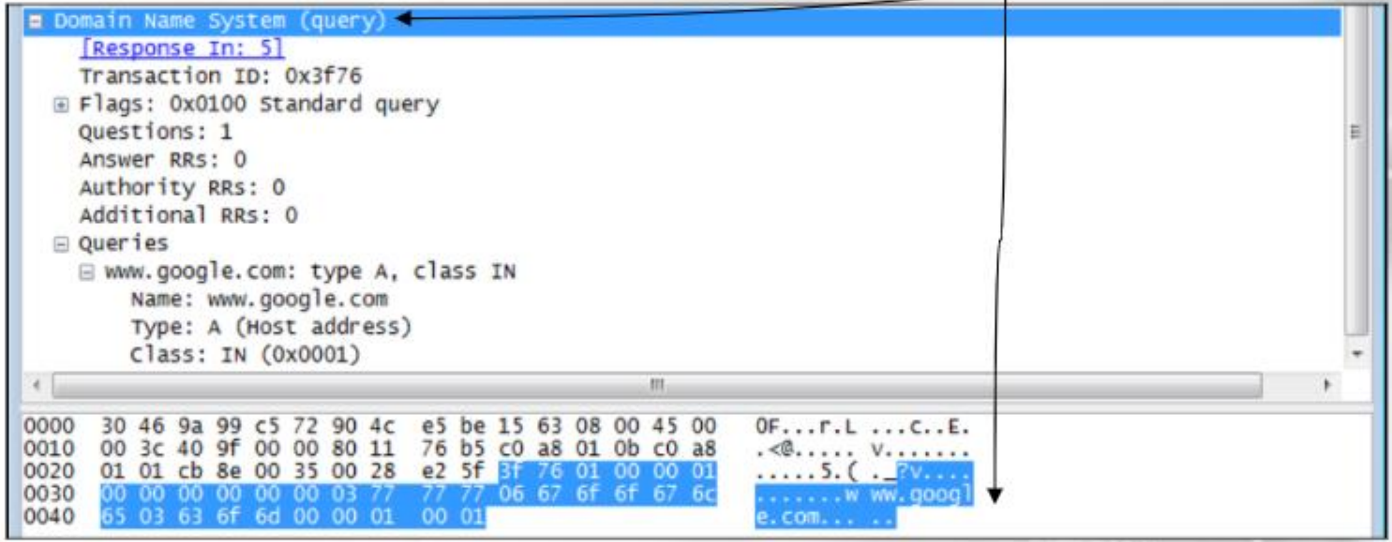
What is important to understand, is that what we see in Wireshark is Wireshark's interpretation. As you can imagine, there is margin for error. Hopefully, Wireshark and other tools feature *packet bytes panes* (displaying the whole frame).

By taking the displayed hexadecimal values in those panes and interpreting them ourselves, we can be 100% certain of the interpretation result.

1.1.2 TCP/IP

Encapsulation

As already mentioned, the packet bytes pane displays the whole frame. We'll have to click on the layer of interest to highlight the values associated with it.



In this case, we are analyzing a DNS query

1.1.3 Request For Comments (RFC)

Knowing how normal traffic looks like is of great importance. Then, and only then, you will be able to spot abnormalities when analyzing traffic.

For this matter, Request For Comments (RFC) is an invaluable resource. RFC are documents that strictly describe the expected standards for a particular protocol.

1.1.3 Request For Comments (RFC)

For example, RFC 793 – Transmission Control Protocol describes TCP functionality and implementation details as well as the interface using which TCP services are requested.

1.1.3 Request For Comments (RFC)

That being said, RFC documents can be ambiguous or even not extensive at times.

As an analyst, you will have to combine RFC with environment context in order to conclude if something poses as a threat or not.

1.1.4 Traffic Analysis Tools

Before we continue diving into the Network Access/Link layer, let's talk about the traffic analysis tools we will utilize.

Wireshark and the *tcpdump* are the weapons of choice when it comes to traffic analysis.

1.1.4 Traffic Analysis Tools

Traffic Analysis Tools

Wireshark is a powerful network protocol analyzer, that can go to the deepest level of packet inspection. It has excellent filtering and searching capabilities.

Wireshark's cons include a sometimes inaccurate interpretation and a difficulty in effectively handling very large pcap files.

1.1.4 Traffic Analysis Tools

Traffic Analysis Tools

tcpdump is also a powerful network protocol analyzer that can go to the deepest level of packet inspection. It can be found on most package managers and it can easily handle very large pcap files.

tcpdump's cons include a minimal protocol decoding list and a requirement for interpreting things ourselves.

OUTLINE

1.2

Analyzing & Detecting IEEE 802.x Link Layer Attacks

1.2 Analyzing & Detecting IEEE 802.x Link Layer Attacks

We briefly covered some important networking concepts, and now, things will get practical. We will start the practical part of this module by analyzing and detecting IEEE 802.x Link layer attacks.

But before analyzing and detecting IEEE 802.x Link layer attacks, let's first have a look at the IEEE 802.x Link layer itself.

1.2.1 The Network Access/Link Layer

The IEEE 802.x Link layers are a family of standards that enable intercommunications between equipment from a variety of manufacturers.

This family of standards actually specifies functions of the physical layer and the data link layer of major LAN protocols.

1.2.1 The Network Access/Link Layer

The most known link layers being used nowadays are:

- 802.3: Ethernet
- 802.11: Wireless
- 802.15.1: Bluetooth

OUTLINE

1.2.1 The Network Access/Link Layer

Let's focus on 802.3 and start covering it, by analyzing the link layer frame.

The link frame actually consists of the Ethernet header plus all following layers.

OUTLINE

1.2.1 The Network Access/Link Layer

After the Ethernet header, we find data whose size could reach up to 1500 bytes. As you already know, that data can be IP, a transport protocol and data (recall encapsulation), but it can also be ARP.

The Ethernet header has a length of 14 bytes. The Ethernet frame, in turn, must have a length of at least 64 bytes, according to specification.

1.2.1 The Network Access/Link Layer

If the frame has a shorter length, a trailer of 0s must be added to pad the number of bytes to 60. Why 60 and not 64? This is because, the Ethernet frame also features a 4-byte trailer known as CRC (Cyclic Redundancy Check), used to identify frame corruption.

If you do the math, the maximum Ethernet frame length is 1518 bytes.

1.2.1 The Network Access/Link Layer

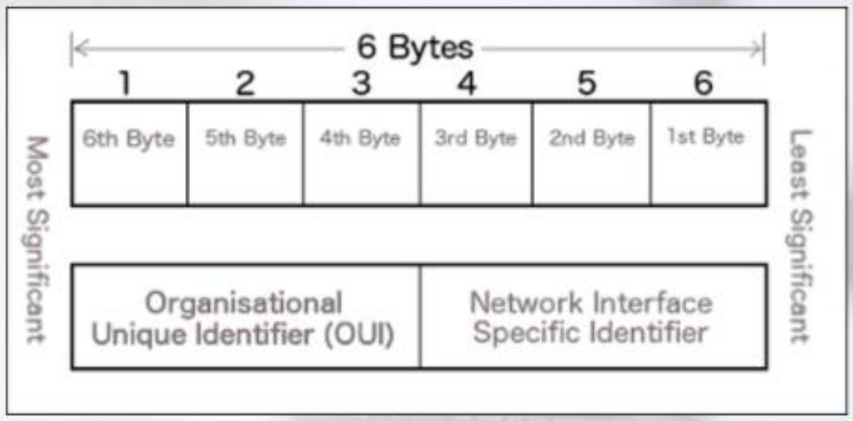
Link layer communication is facilitated by Network Interface Cards (NICs) and the respective device drivers. Using those, a host can interact with the physical medium on which it resides.

Any NIC has a unique identification number known as a MAC address which is issued by the NIC's manufacturer during NIC creation. MAC addresses are static 48-bit long numbers.

1.2.1 The Network Access/Link Layer

Here is a MAC address break down. To decode a MAC address, we split it in half; this will give us the OUI and the network ID.

We can then lookup the OUI on a website, such as https://www.macvendorlookup.com/, to reveal the NICs manufacturer.



https://www.macvendorlookup.com/

OUTLINE

1.2.1 The Network Access/Link Layer

According to the TCP/IP stack, the IP layer will have to find a way to talk to the Link layer; this is done through an IP-to-MAC address association.

The IP layer communicates using IP addresses, whereas the Link layer communicates using MAC addresses.

1.2.1 The Network Access/Link Layer

What makes this association possible in IPv4 is **ARP**.

In IPv6, **Neighbor Solicitation** is used to request for a MAC address associated with a given IPv6 address, and **Neighbor Advertisement** is used for sending the response.

1.2.1 The Network Access/Link Layer

In contrast to MAC addresses, IPv4 addresses are software addresses that can change over time and they have a length of 32 bits.

The same applies for IPv6 addresses, but they have a length of 128 bits.

1.2.1 The Network Access/Link Layer

At this point, it should be noted that ARP traffic is generated only when two hosts residing in same local network want to communicate. If the two hosts reside on different physical segments, traffic will be routed via the Internet layer first and then passed to the Network Access layer.

You can read more about ARP in RFC 826.

https://www.ietf.org/rfc/rfc826.txt

1.2.2 ARP's Security Shortcomings

Over the years ARP has been greatly abused by attackers, due to its inherent security shortcomings.

- There is no way to validate MAC address ownership whenever an ARP request or response is issued.
- ARP is stateless. Whenever an ARP response is received, hosts will create or update a cache entry with the observed IP/MAC pair (regardless of them issuing an ARP request or not).
- An initial ARP request can result in the requester's IP/MAC pair being cached by listening (for broadcasts) hosts; this is done to reduce ARP broadcast requests.

OUTLINE

1.2.2 ARP's Security Shortcomings

Those ARP shortcomings can be leveraged by an attacker to launch man-in-the-middle attacks, where he/she can pose as just another host on the local network or even as a router.

OUTLINE

1.2.2 ARP's Security Shortcomings

If such attacks are executed successfully, the attacker can receive traffic destined for another host, inspect it or alter it and ultimately forward it to the original destination.

Not only that, but the attacker can also receive, inspect or alter the traffic from the original destination and ultimately forward it back to the original sender.

1.2.2 ARP's Security Shortcomings

Let's see how we can detect such attacks on the wire.
First, let's see some normal ARP traffic…

Note: Refer to the Wireshark Display Filter Reference for ARP, here, on more techniques to filter ARP traffic.

https://www.wireshark.org/docs/dfref/a/arp.html

OUTLINE

1.2.2.1 ARP Attacks & Detection

Below is a snapshot of 2 packets: 1 ARP Request & 1 ARP Reply.

arp

No.	Time	Source	Destination	Protocol	Length	Info
11	5.166850	26:11:59:88:53:02	Vmware_a1:f4:d0	ARP	42	Who has 10.54.15.68? Tell 10.54.15.100
12	5.215241	Vmware_a1:f4:d0	26:11:59:88:53:02	ARP	60	10.54.15.68 is at 00:50:56:a1:f4:d0

To see the MAC Address for both the source and destination, we can make a quick change within Wireshark: **View** > **Name Resolution** > **Resolve Physical Addresses**.

arp

No.	Time	Source	Destination	Protocol	Length	Info
11	5.166850	26:11:59:88:53:02	00:50:56:a1:f4:d0	ARP	42	Who has 10.54.15.68? Tell 10.54.15.100
12	5.215241	00:50:56:a1:f4:d0	26:11:59:88:53:02	ARP	60	10.54.15.68 is at 00:50:56:a1:f4:d0

OUTLINE

1.2.2.1 ARP Attacks & Detection

Here we see an ARP Request packet.

We know it's an **ARP Request** by the Opcode, **request (1)**, in the highlighted line. We can see that the device at 10.54.15.100 needs the MAC address for 10.54.15.68 to begin establishing communication with it. If the device at 10.54.15.100 already knew that MAC address for 10.54.15.100, then it would be contained within it's ARP table (`arp -a` or `arp` from the command line).

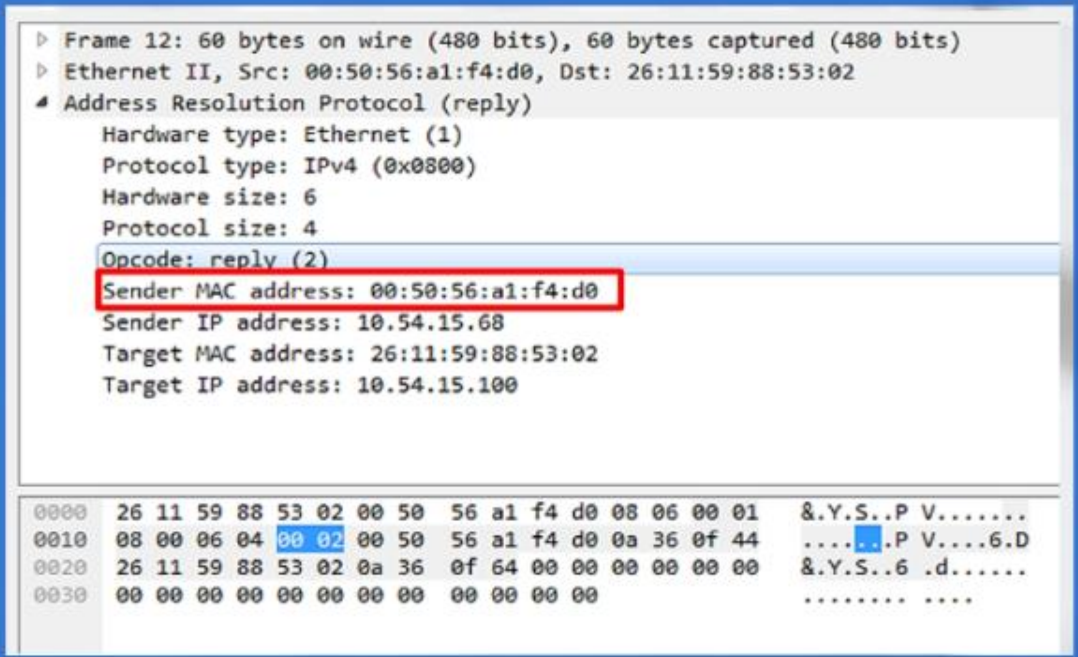
```
▷ Frame 11: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▷ Ethernet II, Src: 26:11:59:88:53:02, Dst: 00:50:56:a1:f4:d0
◢ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 26:11:59:88:53:02
    Sender IP address: 10.54.15.100
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 10.54.15.68

0000  00 50 56 a1 f4 d0 26 11  59 88 53 02 08 06 00 01   .PV...&. Y.S.....
0010  08 00 06 04 00 01 26 11  59 88 53 02 0a 36 0f 64   ......&. Y.S..6.d
0020  00 00 00 00 00 00 0a 36  0f 44                     .......6 .D
```


OUTLINE

1.2.2.1 ARP Attacks & Detection

In this packet, we have the reply to the previous packet.

This packet is the **ARP Reply** packet. We can quickly tell by the Opcode, **reply (2)**. Within this packet, we see that the Sender MAC address has been populated with the MAC address of the device at 10.54.15.68. This MAC address will be added to the ARP table of 10.54.15.100.



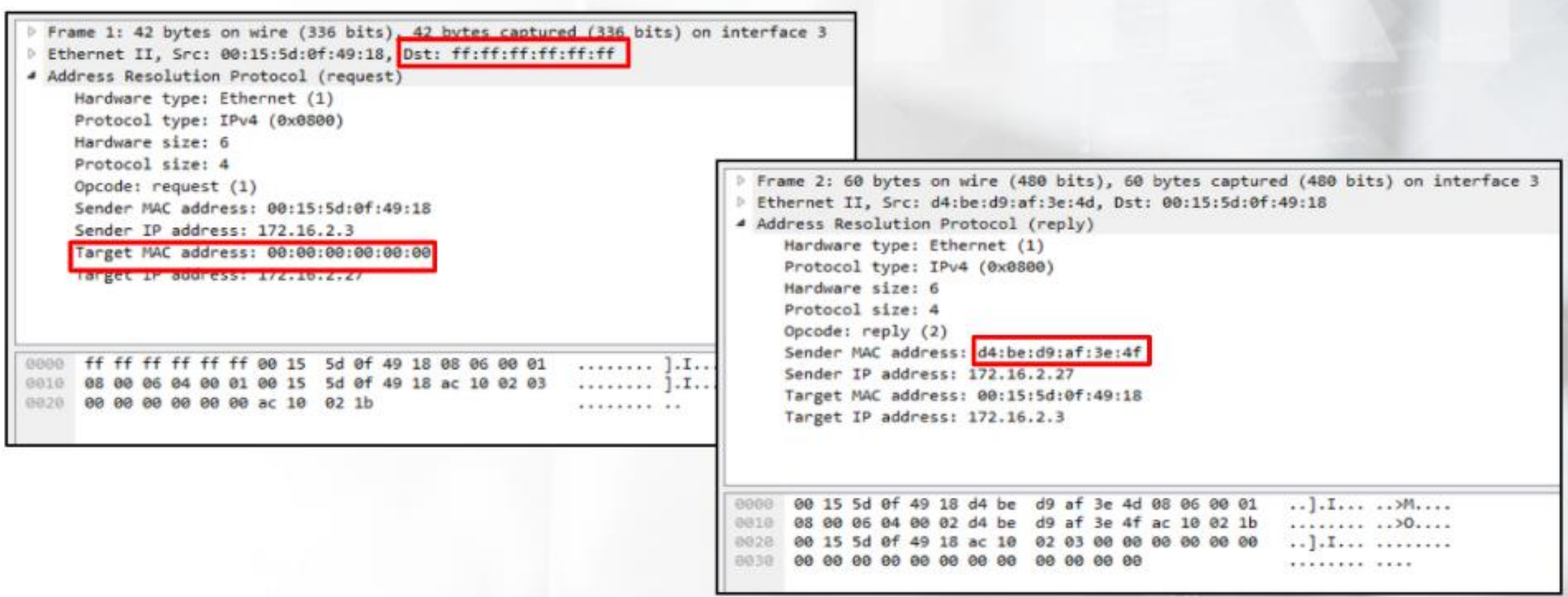
OUTLINE

1.2.2.1 ARP Attacks & Detection

The previous packets were generated within a virtual machine.

The following packet will reflect an ARP Request within a network using a broadcast address as the destination.

1.2.2.1 ARP Attacks & Detection



1.2.2.1 ARP Attacks & Detection

This is how ARP works if one of the hosts in the network asks for it; however, this is not the only way though.

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 reply: It is an ARP reply that has been sent without being requested.

1.2.2.1 ARP Attacks & Detection

Normal Gratuitous ARP

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

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


OUTLINE

1.2.2.1 ARP Attacks & Detection

Attacker-crafted Gratuitous ARP

Looks like a crafted/illegitimate MAC address

```
▼ Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is gratuitous: True]
    Sender MAC address: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)
    Sender IP address: 192.168.11.200 (192.168.11.200)
    Target MAC address: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)
    Target IP address: 192.168.11.200 (192.168.11.200)
```


OUTLINE

1.2.2.1 ARP Attacks & Detection

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.

We will see how shortly.

1.2.2.1 ARP Attacks & Detection

Now, we'll look at examples of ARP traffic using Wireshark.

Remember the following 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.

1.2.2.1 ARP Attacks & Detection

Here we see a snapshot of a packet capture, which shows 7 ARP Request packets sent via broadcast.

Via physical address name resolution within Wireshark, the source device seems to be a Cisco device and it's checking on the status of various devices on the network.

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

OUTLINE

1.2.2.1 ARP Attacks & Detection

Now, how would we know if this is suspicious traffic or not? How do we know it's not just a configuration issue within the Cisco device or it's normal behavior? Do you even have Cisco equipment on the network?

Based on your response to these questions and others, you'll know whether this needs to be looked into further. In our case, we'll assume its normal.

1.2.2.1 ARP Attacks & Detection

How about this traffic? Would it be considered suspicious or normal?

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

OUTLINE

1.2.2.1 ARP Attacks & Detection

That is definitely suspicious traffic.

Even without knowing off hand what legit device can potentially have that MAC address on your network, but just based on the flow and speed of the ARP Requests, we can tell something is odd.

Starting from packet 15, the IP addresses increment by 1 and time intervals between packets are relatively small which indicates a scan.

1.2.2.1 ARP Attacks & Detection

As already mentioned, there are other techniques in which ARP can be used for nefarious purposes, such as **ARP Spoofing/Cache Poisoning** attacks.

1.2.2.1 ARP Attacks & Detection

ARP poisoning can be exploited to add fake information between two communication peers into a local network. In a scenario in which M (the attacker) wants to listen to all the traffic between A and B, M would have to send fake IP/MAC pairs to both A and B, making himself the Man-in-the-Middle.

1.2.2.1 ARP Attacks & Detection

ARP Poisoning

The following are the steps for a successful attack:

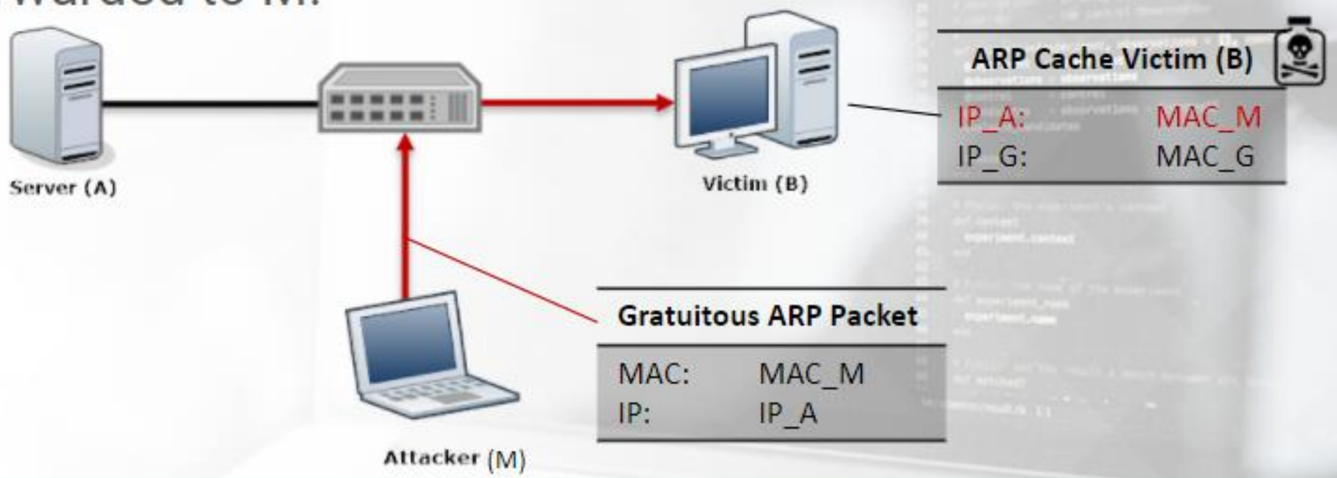
1. M would pretend to be B to A: it will send a gratuitous ARP reply with the pair: IP_B->MAC_M
2. M would pretend to be A to B: it will send a gratuitous ARP reply with the pair: IP_A->MAC_M

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).

1.2.2.1 ARP Attacks & Detection

ARP Poisoning

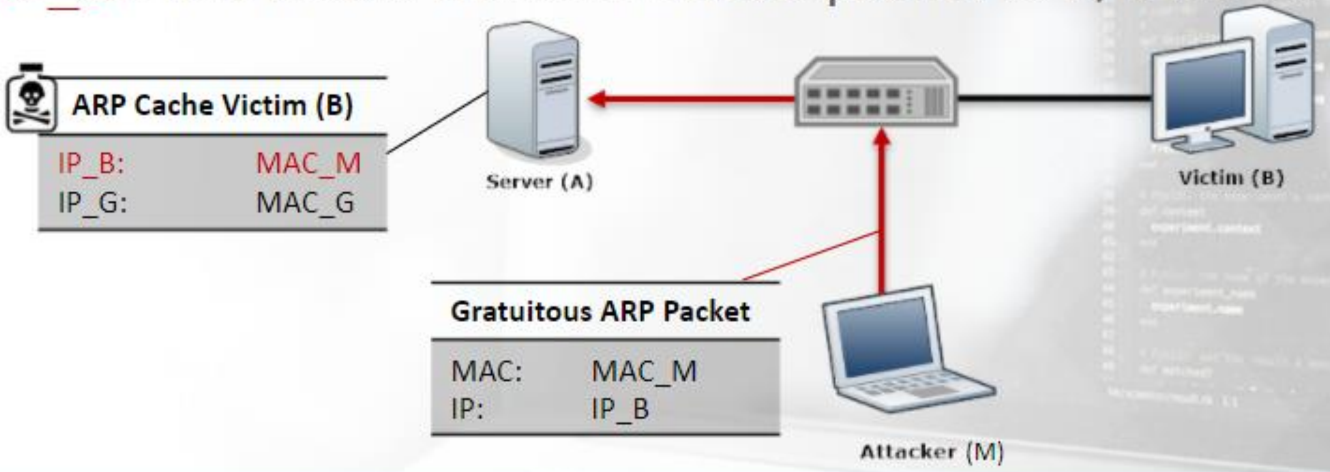
Once the gratuitous ARP packet is sent, B's ARP cache gets poisoned with the entry: `IP_A->MAC_M`. Next time B wants to send a packet to A, it will be forwarded to M.



1.2.2.1 ARP Attacks & Detection

ARP Poisoning

The same thing happens against A. The attacker sends the gratuitous ARP packet, and A's ARP cache gets poisoned with the entry `IP_B->MAC_M`. Next time A wants to send a packet to B, it will be forwarded to M.



1.2.2.1 ARP Attacks & Detection

ARP Poisoning

This attack leaves the MAC address of the attacker in the ARP cache of the victims.

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

1.2.2.1 ARP Attacks & Detection

ARP Poisoning

When a host in a LAN wants to send packets to hosts outside the LAN, it uses the default gateway.

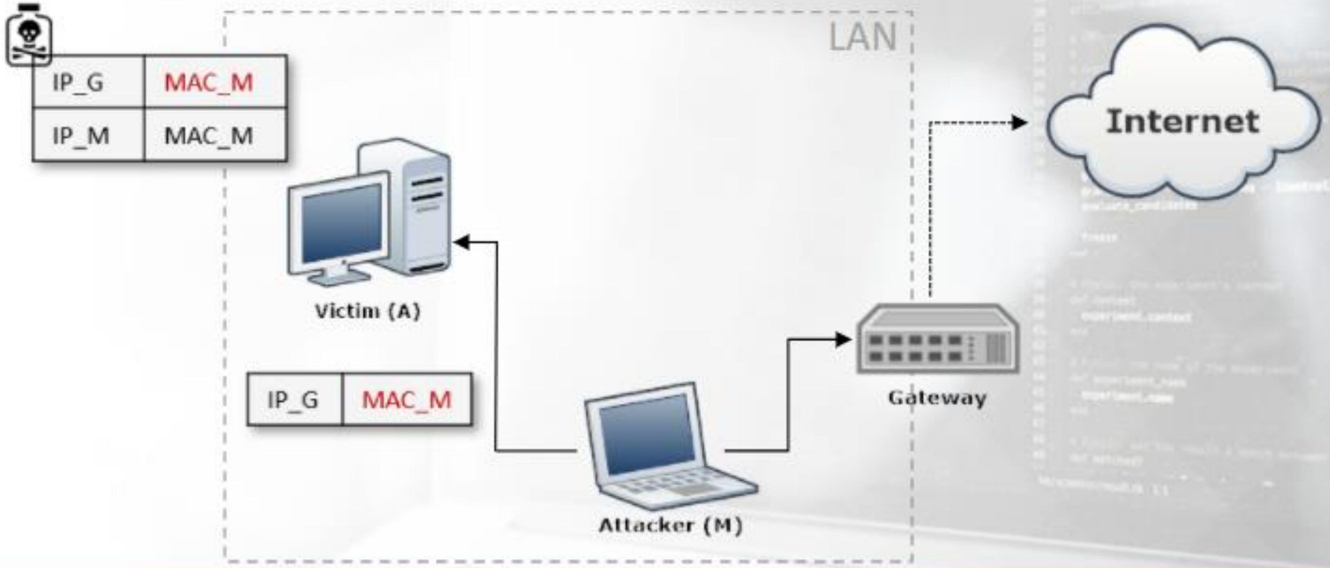
The default gateway MAC address must be used to forward the packet along with the correct IP address configured by the administrator or given by DHCP.

The use of ARP poisoning in this scenario leads to a MITM attack from local to remote.

1.2.2.1 ARP Attacks & Detection

ARP Poisoning

This diagram explains the MitM scenario. Host A sends all the traffic aimed for the internet through the Attacker.



1.2.2.1 ARP Attacks & Detection

ARP Poisoning

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, which will then redirect it to the real gateway.

1.2.2.1 ARP Attacks & Detection

Consider the *arp_poisoning.pcapng file* found in this module's resources. How about this traffic? Would it be considered suspicious or normal?

No.	Time	Source	Destination	Protocol	Length	Version	Info
1	0…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
2	0…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2
5	1…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
6	1…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2

No.	Time	Source	Destination	Protocol	Length	Version	Info
3270	3…	Vmware_eb:f7:91	Vmware_cd:e3:c0	ARP	42		192.168.153.2 is at 00:50:56:eb:f7:91

```
Frame 3270: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: Vmware_eb:f7:91 (00:50:56:eb:f7:91), Dst: Vmware_cd:e3:c0 (00:0c:29:cd:e3:c0)
Address Resolution Protocol (reply)
```

No.	Time	Source	Destination	Protocol	Length	Version	Info
4051	7…	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
4148	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
4149	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2
4270	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
4271	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2
4296	7…	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
4394	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
4395	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2
4537	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
4538	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2
4565	7…	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
4679	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
4680	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2
4821	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.135? Tell 192.168.153.2
4822	7…	Vmware_eb:f7:91	Broadcast	ARP	42		Who has 192.168.153.153? Tell 192.168.153.2
4864	7…	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14

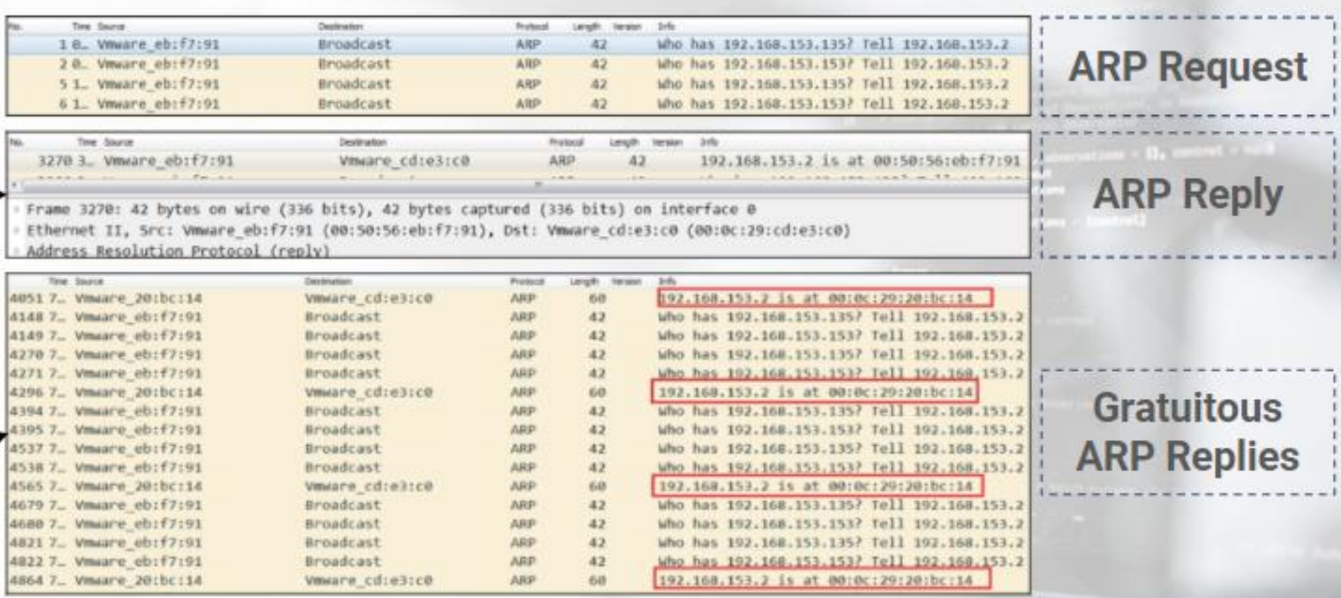
OUTLINE

1.2.2.1 ARP Attacks & Detection

Consider the *arp_poisoning.pcapng file* found in this module's resources. How about this traffic? Would it be considered suspicious or normal?

The router's MAC address is **00:50:56:eb:f7:91**

A malicious device on the network is telling that the router's MAC address is **00:0c:29:20:bc:14**, by issuing gratuitous ARP replies at frequent intervals .



OUTLINE

1.2.2.1 ARP Attacks & Detection

We can actually identify the malicious device on the network since we know the MAC address of the attacker, by doing the following:

Let's filter the traffic looking for frames that hold the attacker's MAC address. We can see the ARP replies we saw earlier in addition to an ACK segment coming from 192.168.153.154 which contains the attacker's MAC address we got earlier.

eth.addr==00:0c:29:20:bc:14

No.	Time	Source	Destination	Protocol	Length	Version	Info
3614	5...	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
3620	5...	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
3625	5...	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
3631	6...	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
3637	6...	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
3643	6...	Vmware_20:bc:14	Vmware_cd:e3:c0	ARP	60		192.168.153.2 is at 00:0c:29:20:bc:14
3647	6...	192.168.153.154	8.41.222.241	TCP	54	4	49530 → 443 [ACK] Seq=1736 Ack=5874 Win=64240 Len=0

OUTLINE

1.2.2.2 ARP Spoofing Prevention

How you can prepare against ARP spoofing attacks, you may ask.

- Using Static ARP could help, but it is not a feasible approach into large and always-changing networks.
- Tools like **arpwatch** can detect but not stop such attacks
- Switches usually feature protections against ARP spoofing

OUTLINE

1.2.3 Other Sniffing Attacks & Detection

With hardware switches, traffic sniffing, in general, is more difficult.

In a normal and not stressed switched network, sniffing for data is impossible, due to the fact that switches store the MAC address to physical switch port pairing in their Content Addressable Memory (CAM) table.

1.2.3 Other Sniffing Attacks & Detection

Efficient attack techniques have been introduced though, to force switches to behave like a hub and then forward frames on all the ports.

1.2.3 Other Sniffing Attacks & Detection

Such a technique is called **MAC Flooding**. MAC flooding is meant to stress the switch and fill its CAM table.

A CAM table keeps all the info required to forward frames to the correct port: `<MAC address - port number - TTL>`.

1.2.3 Other Sniffing Attacks & Detection

When the space in the CAM is filled with fake MAC addresses, the switch cannot learn new MAC addresses.

The only way to keep the network alive is to forward the frames meant to be delivered to the unknown MAC address on all the ports of the switch, thus making it fail open, or act like a Hub.

1.2.3 Other Sniffing Attacks & Detection

That being said, port security measures exist that can 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.

1.2.3 Other Sniffing Attacks & Detection

Consider the *mac_flood.pcapng* file found in this module's resources and assume we are dealing with a switched environment.

How about this traffic? Would it be considered suspicious or normal?

4	5…	178.58.143.30	211.82.179.80	IPv4	60	4
5	5…	173.95.30.91	51.142.110.28	IPv4	60	4
6	5…	20.95.94.114	79.48.168.87	IPv4	60	4
7	5…	44.156.190.13	179.237.151.7	IPv4	60	4
8	5…	59.13.56.57	69.243.240.42	IPv4	60	4
9	5…	158.240.39.109	91.241.192.114	IPv4	60	4
10	5…	38.188.79.23	87.96.115.62	IPv4	60	4
11	5…	103.62.72.92	152.116.235.38	IPv4	60	4
12	5…	143.201.232.5	52.213.160.64	IPv4	60	4
13	5…	102.167.35.67	43.232.243.105	IPv4	60	4
14	5…	251.89.114.49	243.32.172.83	IPv4	60	4
15	5…	170.73.95.91	149.227.104.27	IPv4	60	4
16	5…	67.70.73.106	96.63.103.53	IPv4	60	4
17	5…	76.226.104.58	180.131.3.57	IPv4	60	4
18	5…	74.240.142.56	87.234.171.100	IPv4	60	4
19	5…	100.155.50.114	191.70.228.5	IPv4	60	4
20	5…	158.111.84.82	58.25.177.62	IPv4	60	4
21	5…	238.181.170.32	3.235.83.23	IPv4	60	4

1.2.3 Other Sniffing Attacks & Detection

Consider the *mac_flood.pcapng* file found in this module's resources and assume we are dealing with a switched environment.

How about this traffic? Would it be considered suspicious or normal?

What we can easily identify, is that:

1. The traffic capture file is full of malformed packets.
2. Packets don't belong to the segment where we sniffed traffic (their origin is quite sparse actually).

There must be something going on.

4	5...	178.58.143.30	211.82.179.80	IPv4	60	4
5	5...	173.95.30.91	51.142.110.28	IPv4	60	4
6	5...	20.95.94.114	79.48.168.87	IPv4	60	4
7	5...	44.156.190.13	179.237.151.7	IPv4	60	4
8	5...	59.13.56.57	69.243.240.42	IPv4	60	4
9	5...	158.240.39.109	91.241.192.114	IPv4	60	4
10	5...	38.188.79.23	87.96.115.62	IPv4	60	4
11	5...	103.62.72.92	152.116.235.38	IPv4	60	4
12	5...	143.201.232.5	52.213.160.64	IPv4	60	4
13	5...	102.167.35.67	43.232.243.105	IPv4	60	4
14	5...	251.89.114.49	243.32.172.83	IPv4	60	4
15	5...	170.73.95.91	149.227.104.27	IPv4	60	4
16	5...	67.70.73.106	96.63.103.53	IPv4	60	4
17	5...	76.226.104.58	180.131.3.57	IPv4	60	4
18	5...	74.240.142.56	87.234.171.100	IPv4	60	4
19	5...	100.155.50.114	191.70.228.5	IPv4	60	4
20	5...	158.111.84.82	58.25.177.62	IPv4	60	4
21	5...	238.181.170.32	3.235.83.23	IPv4	60	4

1.2.3 Other Sniffing Attacks & Detection

Let's make sure by utilizing statistical analysis. First, download and install a trial of Colasoft's Capsa. Now open Capsa and navigate to the **Replay** tab. From there, add *mac_flood.pcapng,* choose **Full Analysis** and press **Start**. Once analysis is done, navigate to the **Summary** tab, where you will find the below information.

Address	Count
MAC Address	633,426
IP Address	633,424
Local IP Address	42,191
Remote IP Address	591,233

https://www.colasoft.com/download/products/download_capsa.php

1.2.3 Other Sniffing Attacks & Detection

Let's make sure by utilizing statistical analysis. First, download and install a trial of Colasoft's Capsa. Now open Capsa and navigate to the **Replay** tab. From there, add *mac_flood.pcapng,* choose **Full Analysis** and press **Start**. Once analysis is done, navigate to the **Summary** tab, where you will find the below information.

Address	Count
MAC Address	633,426
IP Address	633,424
Local IP Address	42,191
Remote IP Address	591,233

The amount of unique MAC addresses is unusually high !

OUTLINE

1.2.3 Other Sniffing Attacks & Detection

Let's visualize things to have a better understand of what is going on.

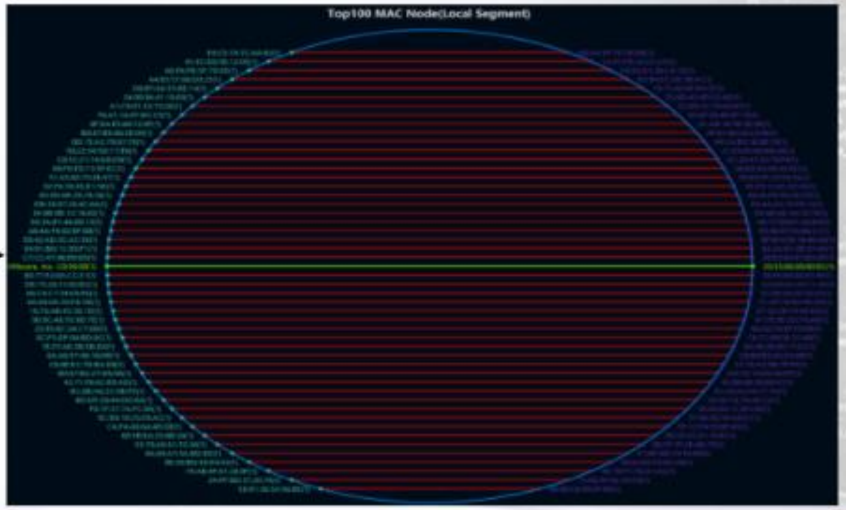
To do so, expand **MAC Explorer** and in turn expand **Local Segment**.

Now, navigate to the **Matrix** tab.

1.2.3 Other Sniffing Attacks & Detection

You will be presented with the following:

Red lines indicate one-way transmitting. It looks like someone is crafting (malformed) packets, in order to perform a MAC flooding attack.



OUTLINE

1.2.3 Other Sniffing Attacks & Detection

For the complete picture, also browse the **MAC Conversation** tab. You will be presented with the following:

Node 1 ->	<- Node 2	Duration	Bytes	Bytes ->	<- Bytes	Packets	Packets ->	<- Packets	First Time Sent ▲	Last Time S
5A:E8:50:1E:7B:E3	16:FF:2F:4C:47:54	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295962000	2017/09/27 05:27:45.295962
40:DF:16:56:36:69	77:AE:F4:0C:CF:5B	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295962000	2017/09/27 05:27:45.295962
43:75:1F:55:28:DF	56:BC:BF:34:70:DA	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295962000	2017/09/27 05:27:45.295962
AD:AC:6B:4B:C0:04	06:C4:29:37:84:B3	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295963000	2017/09/27 05:27:45.295963
59:67:CE:34:2B:82	5A:08:36:7E:3D:8D	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295980000	2017/09/27 05:27:45.295980
5C:91:A1:64:43:84	50:0D:37:20:F7:92	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295981000	2017/09/27 05:27:45.295981
31:A9:2D:6A:87:B0	CE:B3:BB:79:8E:8E	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295981000	2017/09/27 05:27:45.295981
CD:30:AC:4D:28:A7	6E:4C:FA:5A:00:AC	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295981000	2017/09/27 05:27:45.295981
EE:48:5B:73:CB:B0	17:FA:94:20:D3:69	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295981000	2017/09/27 05:27:45.295981
AE:C5:A6:6D:9D:3C	09:E7:EA:28:62:CB	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295982000	2017/09/27 05:27:45.295982
30:08:B8:71:A5:7E	AA:F9:1A:38:DC:09	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295983000	2017/09/27 05:27:45.295983
5E:A3:EE:57:0A:BE	1B:82:07:55:AD:93	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.295999000	2017/09/27 05:27:45.295999
3B:E1:AB:31:8F:F8	AA:52:F4:33:AC:44	00:00:00.000000000	60.00 B	60.00 B	0.00 B	1	1	0	2017/09/27 05:27:45.296000000	2017/09/27 05:27:45.296000

Almost all nodes only send one packet out.
Most packets are 60 bytes.

OUTLINE

1.2.3 Other Sniffing Attacks & Detection

Finally, to be 100% sure of the assumption, browse the **Packet** tab. You will be presented with the following:

No.	Absolute Time	Source	Source Geolocation	Destination	Destination Geolocation	Protocol	Size	Payload	Process	Applicat
195497	05:27:38.071809000	164.31.248.8:5438	Germany	225.102.95.5:4680	Local	TCP	60	0		
195498	05:27:38.071809000	247.87.47.88:12205	Local	183.145.147.51:20592	Shaoxing,Zhejiang,China	TCP	60	0		
195499	05:27:38.071810000	186.113.202.70:6427	Bogotá,Bogota D.C.,Col...	8.149.20.52:50153	United States	TCP	60	0		
195500	05:27:38.071810000	232.67.159.118:41070	Local	221.229.194.92:20421	Nanjing,Jiangsu,China	TCP	60	0		
195501	05:27:38.071811000	77.158.241.94:41646	France	79.186.6.7:44268	Poznan,Greater Poland ...	TCP	60	0		
195502	05:27:38.071811000	244.154.34.91:422	Local	207.135.17.112:36658	Redwood City,California...	TCP	60	0		
195503	05:27:38.071811000	152.122.154.93:43651	Washington,District of ...	52.190.246.118:63648	San Jose,California,Unit...	TCP	60	0		
195504	05:27:38.071812000	225.137.242.79:3581	Local	44.49.150.100:64641	San Diego,California,Uni...	TCP	60	0		
195506	05:27:38.072808000	95.14.99.44:50924	Istanbul,Turkey	152.2.144.114:58693	Chapel Hill,North Caroli...	TCP	60	0		
195508	05:27:38.072809000	12.240.198.121:6439	Middletown Township,...	223.106.254.91:9977	Jiangsu,China	TCP	60	0		
195509	05:27:38.072810000	187.132.136.1:12808	Puebla City,Puebla,Mexi...	253.4.199.38:19760	Local	TCP	60	0		
195510	05:27:38.072810000	27.171.213.71:896	Republic of Korea	216.97.32.93:34280	Dallas,Texas,United States	TCP	60	0		

The time difference between each packet transmission is extremely low, causing pressure to the switch. We can safely conclude we are dealing with a MAC flooding attack.

1.2.4 802.11 Wireless

To conclude covering the Network Access/Link Layer, let's also take a look at 802.11 wireless.

We will focus on a clear-text frame that can be provide us with some context.

OUTLINE

1.2.4 802.11 Wireless

Compared to wired packets, wireless ones feature a 802.11 (layer 2) header. This header contains additional information regarding the packet and medium upon which it travels. 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

  subtypes
- **Control**: Delivery of packets is enabled by those packets. Congestion is also "regulated" by them.
  - Request-to-send packets
  - Clear-to-send packets

  subtypes
- **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.

1.2.4 802.11 Wireless

The structure of a wireless packet depends on its type and subtype. As you can imagine, a lot of different structures exist. Let's focus on one of the most informative ones, the beacon packet.

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

1.2.4 802.11 Wireless

Beacon frame capture

The 802.11 management frame header contains information such as:

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

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


OUTLINE

1.3

Analyzing & Detecting IP Layer Attacks

OUTLINE

1.3 Analyzing & Detecting IP Layer Attacks

Now that we have covered analyzing and detecting IEEE 802.x Link layer attacks, it is time to focus our attention on the IP layer.

But before analyzing and detecting IP layer attacks, let's first have a look at the IP layer itself.

1.3.1 The IP Layer

The IP layer's functions are related with how packets are transferred from one hop to another.

Source and destination IP addresses are used by the IP layer for inter-host communication. IP addresses reside in the IP header of the IP packet.

1.3.1 The IP Layer

It should be noted that IP packets travel individually as they are directed to their destination.

For example, IP packets from the same source that are travelling to the same destination could get there through different routes.

1.3.1 The IP Layer

During packet delivery, a lot of things can go wrong. The IP layer has no built-in mechanism to identify when a packet gets lost, expired or dropped.

The Transport protocol or the application itself is responsible for identifying and resolving any packet loss.

You can read more about IP in RFC 791.

We will cover both IPv4 and IPv6.

https://www.ietf.org/rfc/rfc791.txt

OUTLINE

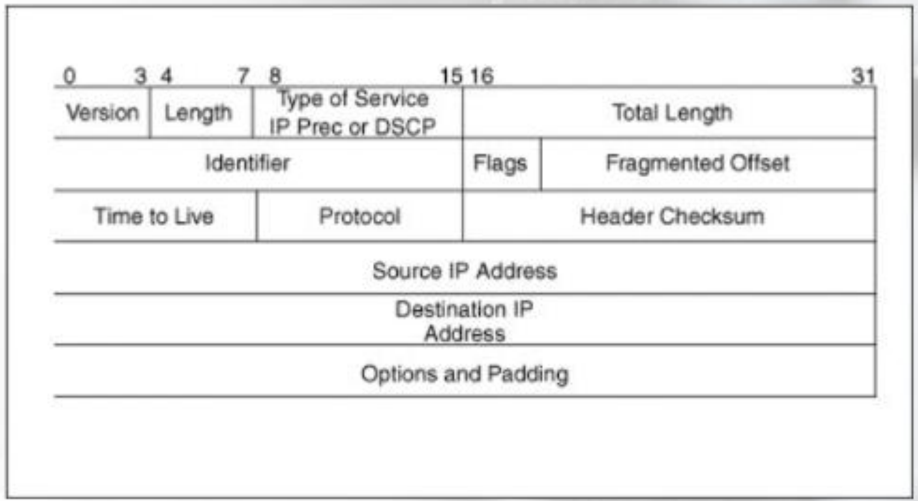
1.3.1 The IP Layer

Below you can see a representation of the IP header. Note that in order to facilitate de-encapsulation, the IP header features three distinct length values.

Length (IP header length): A field containing the length of the IP header.

Total Length (IP datagram/packet length): Specifies the length of the IP packet that includes the IP header and the user data.

Fragment(ed) Offset: In case of a packet being divided, the fragmentation offset value will be used to reassemble the packet.



OUTLINE

1.3.2 Important IPv4 Fields

Let's elaborate more on some important and, oftentimes, abused IPv4 fields.

First, an IP **Version** field exists indicating whether we are dealing with IPv4 or IPv6 (valid version numbers are 4 and 6 only). If an invalid IP Version value is identified at the host or router level, the datagram/packet must be silently discarded according to the respective RFC.

1.3.2 Important IPv4 Fields

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

Consider the *bad_ip_version.pcap* file found in this module's resources. How about this traffic? Would it be considered suspicious or normal?

```
>> tcpdump -r bad_ip_version.pcap -ntx
```

```
# tcpdump -r bad_ip_version.pcap -ntx
reading from file bad_ip_version.pcap, link-type EN10MB (Ethernet)
IP7
        0x0000:  750c 04db 0001 0000 a0bd 61fe c0a8 0106
        0x0010:  c0a8 0104 fd64 fe8d 4dec 5b7c dc3b e405
        0x0020:  a5ad 831f c7b8 7df7 4f13 3a86 eec0 d334
        0x0030:  5020 3269 f56b 2889 d04c 22c0 5c59 6683
        0x0040:  42b0 68c9 0678 ff6e 27c1 188b 7603 037d
        0x0050:  df5f 7368 020b 2c4c cfe4 de46 1052 7492
```

1.3.2 Important IPv4 Fields

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

Consider the *bad_ip_version.pcap* file found in this module's resources. How about this traffic? Would it be considered suspicious or normal?

```
>> tcpdump -r bad_ip_version.pcap -ntx
```

In yellow, we can see that the IP version field has a value of 0x7.

We can safely conclude we are dealing with malicious traffic.

```
# tcpdump -r bad_ip_version.pcap -ntx
reading from file bad_ip_version.pcap, link-type EN10MB (Ethernet)
IP7
        0x0000:  750c 04db 0001 0000 a0bd 61fe c0a8 0106
        0x0010:  c0a8 0104 fd64 fe8d 4dec 5b7c dc3b e405
        0x0020:  a5ad 831f c7b8 7df7 4f13 3a86 eec0 d334
        0x0030:  5020 3269 f56b 2889 d04c 22c0 5c59 6683
        0x0040:  42b0 68c9 0678 ff6e 27c1 188b 7603 037d
        0x0050:  df5f 7368 020b 2c4c cfe4 de46 1052 7492
```


OUTLINE

1.3.2 Important IPv4 Fields

Let's continue analyzing important IPv4 header fields.

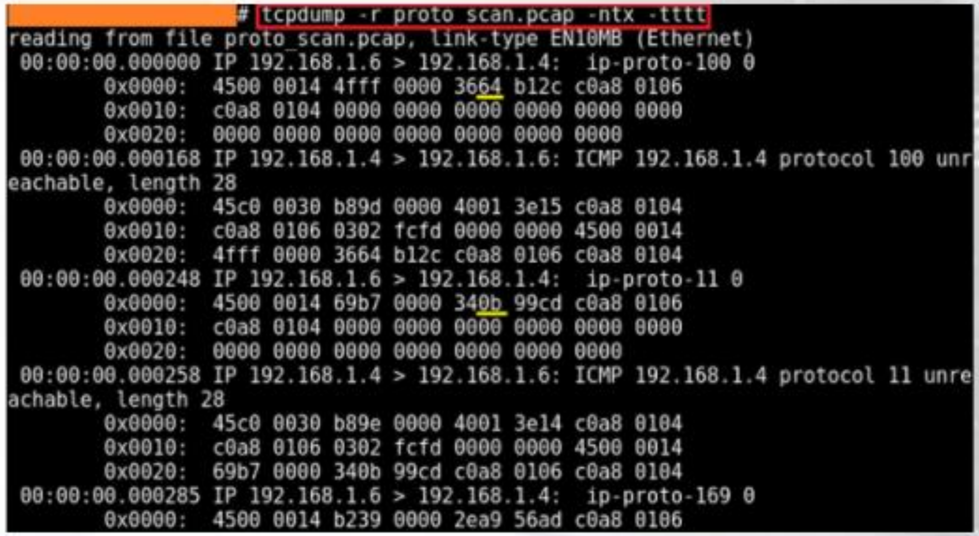
This time, let's look at the **IPv4 Protocol Number**. Depending on this field's value the protocol/transport layer that follows the IP header is identified.

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.

1.3.2 Important IPv4 Fields

Consider the *proto_scan.pcap* file found in this module's resources. How about this traffic? Would it be considered suspicious or normal?

```
>> tcpdump -r proto_scan.pcap -ntx -tttt
```



```
[redacted]# tcpdump -r proto_scan.pcap -ntx -tttt
reading from file proto_scan.pcap, link-type EN10MB (Ethernet)
 00:00:00.000000 IP 192.168.1.6 > 192.168.1.4:  ip-proto-100 0
        0x0000:  4500 0014 4fff 0000 3664 b12c c0a8 0106
        0x0010:  c0a8 0104 0000 0000 0000 0000 0000 0000
        0x0020:  0000 0000 0000 0000 0000 0000 0000
 00:00:00.000168 IP 192.168.1.4 > 192.168.1.6: ICMP 192.168.1.4 protocol 100 unr
eachable, length 28
        0x0000:  45c0 0030 b89d 0000 4001 3e15 c0a8 0104
        0x0010:  c0a8 0106 0302 fcfd 0000 0000 4500 0014
        0x0020:  4fff 0000 3664 b12c c0a8 0106 c0a8 0104
 00:00:00.000248 IP 192.168.1.6 > 192.168.1.4:  ip-proto-11 0
        0x0000:  4500 0014 69b7 0000 340b 99cd c0a8 0106
        0x0010:  c0a8 0104 0000 0000 0000 0000 0000 0000
        0x0020:  0000 0000 0000 0000 0000 0000 0000
 00:00:00.000258 IP 192.168.1.4 > 192.168.1.6: ICMP 192.168.1.4 protocol 11 unre
achable, length 28
        0x0000:  45c0 0030 b89e 0000 4001 3e14 c0a8 0104
        0x0010:  c0a8 0106 0302 fcfd 0000 0000 4500 0014
        0x0020:  69b7 0000 340b 99cd c0a8 0106 c0a8 0104
 00:00:00.000285 IP 192.168.1.6 > 192.168.1.4:  ip-proto-169 0
        0x0000:  4500 0014 b239 0000 2ea9 56ad c0a8 0106
```


OUTLINE

1.3.2 Important IPv4 Fields

Consider the *proto_scan.pcap* file found in this module's resources. How about this traffic? Would it be considered suspicious or normal?

```
>> tcpdump -r proto_scan.pcap -ntx -tttt
```

1. It is obvious that the 192.168.1.6 host sends a great number of packets to the 192.168.1.4 one at very at short intervals.

2. The first IPv4 packet contains the 64 value hexadecimal (GMTP protocol) in its IP Protocol field. The third IPv4 packet contains the 0b hexadecimal value (Network Voice protocol) in its IP Protocol field. The same pattern continues with all upcoming IPv4 packets containing all kinds of protocols in their IP Protocol fields and being sent at short intervals.

3. The multiple `ICMP protocol [id] unreachable` messages are a known indicator, based on which Nmap concludes if a protocol is supported on a host or not.

We are most probably dealing with a malicious IP Protocol Nmap scan

```
[redacted]# tcpdump -r proto_scan.pcap -ntx -tttt
reading from file proto_scan.pcap, link-type EN10MB (Ethernet)
 00:00:00.000000 IP 192.168.1.6 > 192.168.1.4:  ip-proto-100 0
        0x0000:  4500 0014 4fff 0000 3664 b12c c0a8 0106
        0x0010:  c0a8 0104 0000 0000 0000 0000 0000 0000
        0x0020:  0000 0000 0000 0000 0000 0000 0000
 00:00:00.000168 IP 192.168.1.4 > 192.168.1.6: ICMP 192.168.1.4 protocol 100 unr
eachable, length 28
        0x0000:  45c0 0030 b89d 0000 4001 3e15 c0a8 0104
        0x0010:  c0a8 0106 0302 fcfd 0000 0000 4500 0014
        0x0020:  4fff 0000 3664 b12c c0a8 0106 c0a8 0104
 00:00:00.000248 IP 192.168.1.6 > 192.168.1.4:  ip-proto-11 0
        0x0000:  4500 0014 69b7 0000 340b 99cd c0a8 0106
        0x0010:  c0a8 0104 0000 0000 0000 0000 0000 0000
        0x0020:  0000 0000 0000 0000 0000 0000 0000
 00:00:00.000258 IP 192.168.1.4 > 192.168.1.6: ICMP 192.168.1.4 protocol 11 unre
achable, length 28
        0x0000:  45c0 0030 b89e 0000 4001 3e14 c0a8 0104
        0x0010:  c0a8 0106 0302 fcfd 0000 0000 4500 0014
        0x0020:  69b7 0000 340b 99cd c0a8 0106 c0a8 0104
 00:00:00.000285 IP 192.168.1.6 > 192.168.1.4:  ip-proto-169 0
        0x0000:  4500 0014 b239 0000 2ea9 56ad c0a8 0106
```


OUTLINE

1.3.2 Important IPv4 Fields

Another important set of IPv4 header fields are the **Source IP Address** and the **Destination IP Address** fields. There are three golden detection rules that are related to this set of IPv4 header fields.

- 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.
- 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.
- Private network addresses or the loopback mode address also require your attention

https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml

1.3.2.1 Abusing Fragmentation & Detection

Let's talk briefly about fragmentation. Fragmentation is the action of dividing a packet whose size is greater than the Maximum Transmission Unit (MTU) into equal-sized (except for the last) packets, whose size is less or equal to the MTU. Fragmentation 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.

1.3.2.1 Abusing Fragmentation & Detection

What is of interested to us, is that fragmentation is oftentimes abused for IDS/IPS evasion purposes.

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.

1.3.2.1 Abusing Fragmentation & Detection

That being said, 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

1.3.2.1 Abusing Fragmentation & Detection

For the IDS/IPS to safely perform such packet reassembling and inspection, it should act just like the destination host does.

Let's consider the delayed fragments case. If the IDS/IPS, due to performance limitations, doesn't wait as long as the destination does for a fragment to arrive, a delayed fragment containing a malicious payload could evade it and exploit the destination.

1.3.2.1 Abusing Fragmentation & Detection

Consider the *frag-based_scan.pcap* file found in this module's resources. Do you think this traffic is malicious?

```
>> tcpdump -r frag-based_scan.pcap -ntx -tttt -v
```

```
[redacted]# tcpdump -r frag-based scan.pcap -ntx -tttt -v
reading from file frag-based_scan.pcap, link-type EN10MB (Ethernet)
 00:00:00.000000 IP (tos 0x0, ttl 64, id 47486, offset 0, flags [+], proto ICMP (1), length 28)
    192.168.1.6 > 192.168.1.4: ICMP echo request, id 33286, seq 0, length 8
        0x0000:  4500 001c b97e 2000 4001 1e08 c0a8 0106
        0x0010:  c0a8 0104 0800 75f9 8206 0000 0000 0000
        0x0020:  0000 0000 0000 0000 0000 0000 0000
 00:00:31.881589 IP (tos 0xc0, ttl 64, id 39831, offset 0, flags [none], proto ICMP (1), length 56)
    192.168.1.4 > 192.168.1.6: ICMP ip reassembly time exceeded, length 36
        IP (tos 0x0, ttl 64, id 47486, offset 0, flags [+], proto ICMP (1), length 28)
    192.168.1.6 > 192.168.1.4: ICMP echo request, id 33286, seq 0, length 8
        0x0000:  45c0 0038 9b97 0000 4001 5b13 c0a8 0104
        0x0010:  c0a8 0106 0b01 f4fe 0000 0000 4500 001c
```


OUTLINE

1.3.2.1 Abusing Fragmentation & Detection

Consider the *frag-based_scan.pcap* file found in this module's resources. Do you think this traffic is malicious?

```
>> tcpdump -r frag-based_scan.pcap -ntx -tttt -v
```

Fragment ID

Fragment offset

Fragments follow

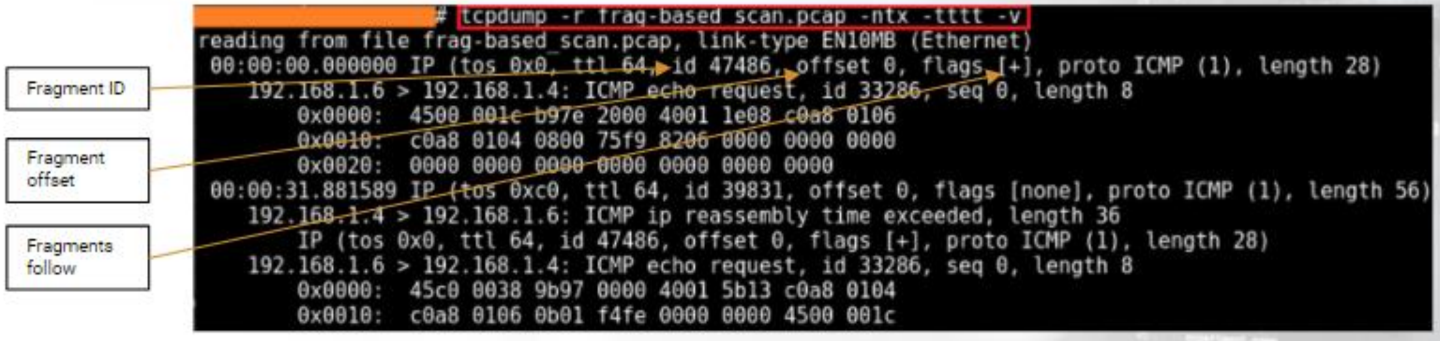
```
[redacted]# tcpdump -r frag-based scan.pcap -ntx -tttt -v
reading from file frag-based_scan.pcap, link-type EN10MB (Ethernet)
 00:00:00.000000 IP (tos 0x0, ttl 64, id 47486, offset 0, flags [+], proto ICMP (1), length 28)
    192.168.1.6 > 192.168.1.4: ICMP echo request, id 33286, seq 0, length 8
        0x0000:  4500 001c b97e 2000 4001 1e08 c0a8 0106
        0x0010:  c0a8 0104 0800 75f9 8206 0000 0000 0000
        0x0020:  0000 0000 0000 0000 0000 0000 0000
 00:00:31.881589 IP (tos 0xc0, ttl 64, id 39831, offset 0, flags [none], proto ICMP (1), length 56)
    192.168.1.4 > 192.168.1.6: ICMP ip reassembly time exceeded, length 36
        IP (tos 0x0, ttl 64, id 47486, offset 0, flags [+], proto ICMP (1), length 28)
    192.168.1.6 > 192.168.1.4: ICMP echo request, id 33286, seq 0, length 8
        0x0000:  45c0 0038 9b97 0000 4001 5b13 c0a8 0104
        0x0010:  c0a8 0106 0b01 f4fe 0000 0000 4500 001c
```


OUTLINE

1.3.2.1 Abusing Fragmentation & Detection

Consider the *frag-based_scan.pcap* file found in this module's resources. Do you think this traffic is malicious?

```
>> tcpdump -r frag-based_scan.pcap -ntx -tttt -v
```



HINT: A stealthy mapping technique exists that leverages fragmentation. Specifically, the attacker sends an incomplete set of fragments (the zero offset fragment must be included) to the scanned hosts. Of course, the scanned hosts should be listening on the specified port or protocol.

If this is the case, once the first fragment is received, a timer is set. Once this timer expires, the receiving host transmits an ICMP "`Fragment reassembly time exceeded`" error back to the attacker.

OUTLINE

1.3.2.1 Abusing Fragmentation & Detection

Consider the *frag-based_scan.pcap* file found in this module's resources. Do you think this traffic is malicious?

```
>> tcpdump -r frag-based_scan.pcap -ntx -tttt -v
```

1. Although there is a flag designating that fragments will follow, we see no fragments arriving for the next 30 seconds

2. Ultimately, we notice an ICMP "reassembly time exceeded" error being sent back to the source that transmitted the incomplete fragment.

We are either dealing with a host misconfiguration or a stealthy scanning attempt.

```
[redacted]# tcpdump -r frag-based_scan.pcap -ntx -tttt -v
reading from file frag-based_scan.pcap, link-type EN10MB (Ethernet)
 00:00:00.000000 IP (tos 0x0, ttl 64, id 47486, offset 0, flags [+], proto ICMP (1), length 28)
    192.168.1.6 > 192.168.1.4: ICMP echo request, id 33286, seq 0, length 8
        0x0000:  4500 001c b97e 2000 4001 1e08 c0a8 0106
        0x0010:  c0a8 0104 0800 75f9 8206 0000 0000 0000
        0x0020:  0000 0000 0000 0000 0000 0000 0000
 00:00:31.881589 IP (tos 0xc0, ttl 64, id 39831, offset 0, flags [none], proto ICMP (1), length 56)
    192.168.1.4 > 192.168.1.6: ICMP ip reassembly time exceeded, length 36
        IP (tos 0x0, ttl 64, id 47486, offset 0, flags [+], proto ICMP (1), length 28)
    192.168.1.6 > 192.168.1.4: ICMP echo request, id 33286, seq 0, length 8
        0x0000:  45c0 0038 9b97 0000 4001 5b13 c0a8 0104
        0x0010:  c0a8 0106 0b01 f4fe 0000 0000 4500 001c
```

1.3.2.1 Abusing Fragmentation & Detection

Let's see more malicious fragmentation examples.

An old DoS technique was sending an IP packet exceeding the 65535 bytes limit of data via a ping command. As you can imagine, this overly big packet would be fragmented and reassembled at the destination host. When older Operating Systems tried to reassemble such a packet they experienced system crashes, reboots or major degradations in performance.

1.3.2.1 Abusing Fragmentation & Detection

In a traffic capture of such an attack, you should see the following moments before the vulnerable system crashed.

This attack is known as ping-of-death.

```
        0x05b0:  4141 4141 4141 4141 4141 4141 4141 4141
        0x05c0:  4141 4141 4141 4141 4141 4141 4141 4141
        0x05d0:  4141 4141 4141 4141 4141 4141
00:00:00.032431 IP (tos 0x0, ttl 64, id 21223, offset 65120, flags [+], proto ICMP (1), length 1500)
```

Right after that, the value of the reassembled packet will exceed the 65535 bytes limit.

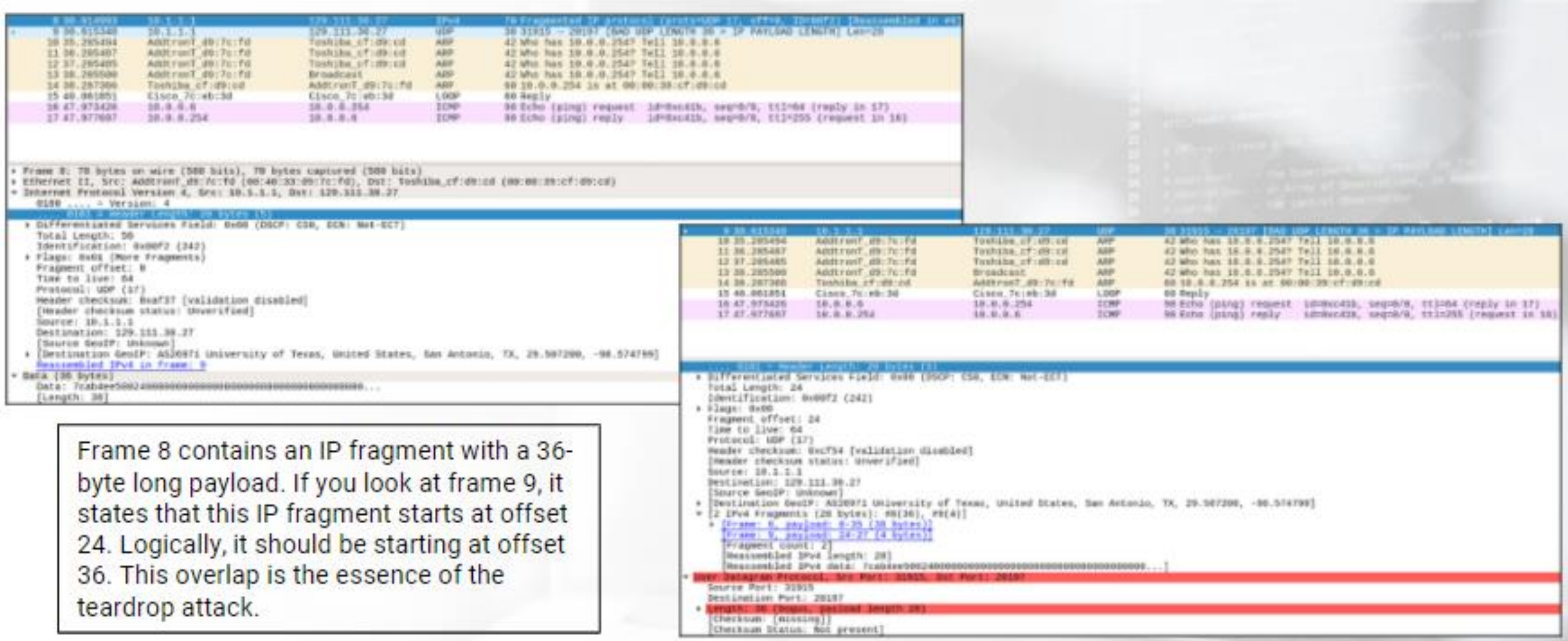
1.3.2.1 Abusing Fragmentation & Detection

Another malicious fragmentation example is the old and UDP-based Teardrop attack.

In this case, crafted overlapping fragments were utilized in order to introduce ambiguities in the reassembling procedure and cause vulnerable systems to crash.

1.3.2.1 Abusing Fragmentation & Detection

Teardrop attack on the wire



Frame 8 contains an IP fragment with a 36-byte long payload. If you look at frame 9, it states that this IP fragment starts at offset 24. Logically, it should be starting at offset 36. This overlap is the essence of the teardrop attack.

OUTLINE

1.3.2.1 Abusing Fragmentation & Detection

You may think that there is no real value in studying older/patched attacks, but older attack mechanics are constantly being updated and reused to discover new vulnerabilities.

This was the case with CVE-2018-5391 (aka FragmentSmack). This attack resembled the way the original Teardrop attack was executed and affected a plethora of modern Windows (and Linux) targets.

https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/ADV180022

1.3.3 IPv6

Truth be told, when the IPv4 specification was written, no one could have predicted the amount of Internet-connected devices we have today.

We are currently experiencing the exhaustion of the IPv4 address space. For this reason, a new version of the IP specification was created back in 1998. This new version of the IP specification was IPv6.

1.3.3 IPv6

This is only part of the story.

IPv6 also features security enhancements and higher packet size limits.

1.3.3 IPv6

IPv4 addresses are 32-bits long, whereas IPv6 addresses are 128-bits long. This fact makes IPv6 addresses a bit difficult to manage. We usually come across IPv6 addresses being written in eight groups of 2 bytes in hexadecimal (e.g.1111:aaaa:2222:bbbb:3333:cccc:4444:dddd)

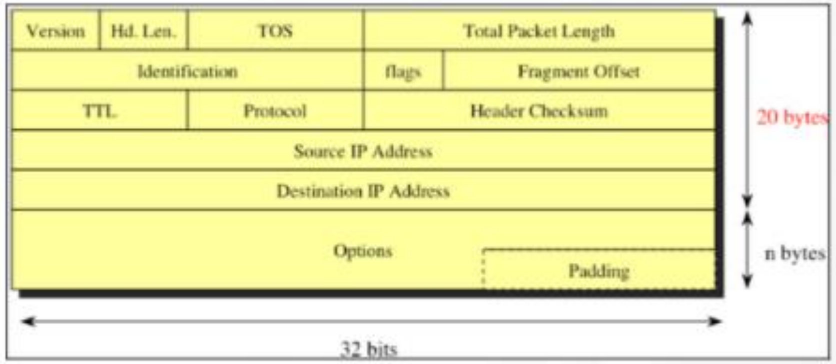
More on IPv6 addresses can be found at the following resource:
http://www.gestioip.net/docu/ipv6_address_examples.html

OUTLINE

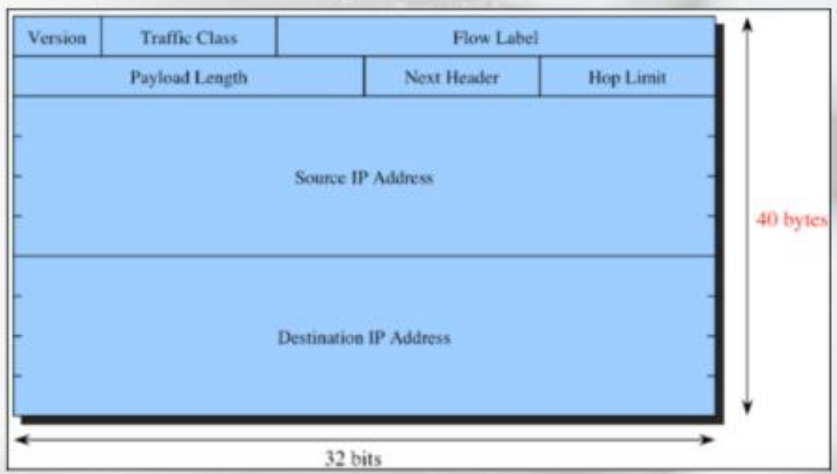
1.3.3 IPv6

Let's also see a brief comparison between the IPv4 and IPv6 headers.

IPv4 Header



IPv6 Header



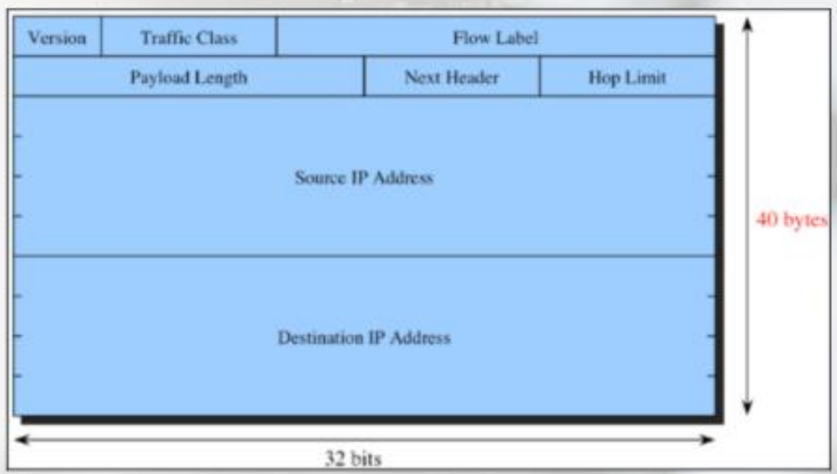
OUTLINE

1.3.3 IPv6

Let's also see a brief comparison between the IPv4 and IPv6 headers.

Version	The version field is 4 bits long and contains the IP version to be expected in the following contents; since we are talking about IPv6, this value is always going to be 6 (0110).
Traffic Class	The traffic class field is 8 bits long and operates the same as the IPv4 Type of Service field; this includes support for the marking of traffic based on a differentiated services code point (DSCP).
Flow Label	The flow label field is 20 bits long and is new to IPv6. It enables the ability to track specific traffic flows at the network layer.
Payload Length	The payload length field is 16 bits long and operates the same as the IPv4 length field; this field includes the length of the data portion of the IPv6 packet.
Next Header	The next header field is 8 bits long and operates similarly to the IPv4 protocol field. The next header field indicates what to expect after the basic IPv6 header; this includes options like a TCP or UDP header and packet.
Hop Limit	The hop limit field is 8 bits long and operates similarly to the IPv4 Time to Live field. This field is used to specify the maximum number of routers that the packet is allowed to travel through before being discarded.
Source Address	The source address field is 128 bits long and operates the same as the IPv4 source address field, with the exception of the length differences.
Destination Address	The destination address field is 128 bits long and operates the same as the IPv4 destination address field, with the exception of the length differences.

IPv6 Header



OUTLINE

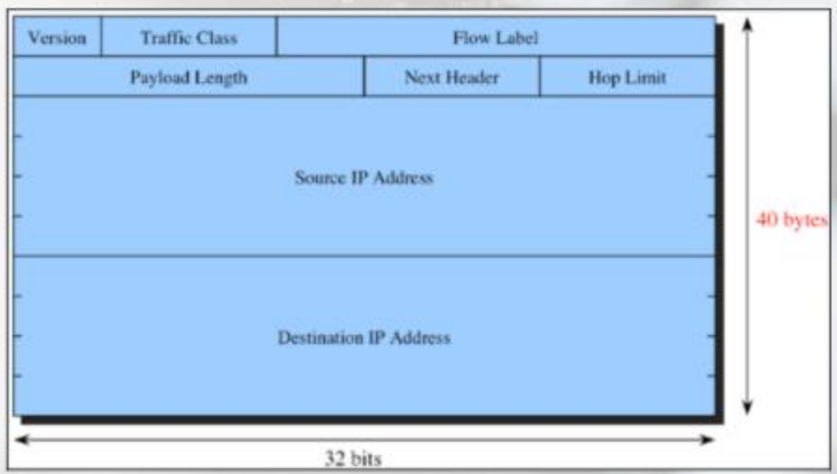
1.3.3 IPv6

Let's also see a brief comparison between the IPv4 and IPv6 headers.

- Any IP options can now be specified in extension headers between the IPV6 header and the transport-related portion of the packet.
- Fragmentation, Flags, Checksum, etc. can be specified in another extension header.
- Validation now lies in the shoulders of the protocol checksums and their pseudo-headers. This means no re-computing of the IPv6 header is required by the routers.

Version	The version field is 4 bits long and contains the IP version to be expected in the following contents; since we are talking about IPv6, this value is always going to be 6 (0110).
Traffic Class	The traffic class field is 8 bits long and operates the same as the IPv4 Type of Service field; this includes support for the marking of traffic based on a differentiated services code point (DSCP).
Flow Label	The flow label field is 20 bits long and is new to IPv6. It enables the ability to track specific traffic flows at the network layer.
Payload Length	The payload length field is 16 bits long and operates the same as the IPv4 length field; this field includes the length of the data portion of the IPv6 packet.
Next Header	The next header field is 8 bits long and operates similarly to the IPv4 protocol field. The next header field indicates what to expect after the basic IPv6 header; this includes options like a TCP or UDP header and packet.
Hop Limit	The hop limit field is 8 bits long and operates similarly to the IPv4 Time to Live field. This field is used to specify the maximum number of routers that the packet is allowed to travel through before being discarded.
Source Address	The source address field is 128 bits long and operates the same as the IPv4 source address field, with the exception of the length differences.
Destination Address	The destination address field is 128 bits long and operates the same as the IPv4 destination address field, with the exception of the length differences.

IPv6 Header



1.3.3.1 Important IPv6 Fields

Let's now cover some important, and oftentimes, abused fields of the IPv6 header.

For more details on the IPv6 header, please refer to RFC 2460.

https://tools.ietf.org/html/rfc2460#page-4

1.3.3.1 Important IPv6 Fields

An important set of IPv6 header fields are the **Source Address** and the **Destination Address** fields. There are three golden detection rules that are related with this set of IPv6 header fields.

- Incoming traffic to your network should obviously have a Source Address that **doesn't** belong to your network address space. If it does, it is most probably crafted. Incoming traffic with Destination Address being a multicast/anycast address needs investigation.
- 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. Outgoing traffic with Destination Address being a multicast/anycast address needs investigation.
- Private network addresses also require your attention.

https://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml

1.3.3.1 Important IPv6 Fields

Another important set of IPv6 header fields are the **Traffic Class** and the **Flow Label** fields. Each one of them (or both at the same time) can be used by an attacker to establish a covert channel of communication.

The following values should be expected:

- Traffic Class: 0 (unless QoS is used)
- Flow Label: 0

1.3.3.1 Important IPv6 Fields

As previously mentioned, the IP Options portion of the IPv4 header has moved into the Extension Header (EH) of the IPv6 header.

No.	Name	Functions	Remarks
0	Hop-by-Hop Options	Carries options for hops, e.g. Router Alert (for MLD, RSVP)	Must be examined by every hop on the path. Must be first EH, only one allowed per packet.
60	Destination Options	Carries options for destination (e.g. for Mobile IPv6)	Processed by destination node only.
43	Routing Header	Lists IPv6 nodes that must be "hopped" on the way to destination	Different types, partly deprecated (RFC 5095), Mobile IP (RFC 6275).
44	Fragmentation Header	Fragmentation (at source)	Fragmentation (at source)

Other examples: 6:TCP, 17:UDP, 58:ICMPv6, 50/51: ESP/AH (IPSec)

Source: FIRST/SWITCH

1.3.3.1 Important IPv6 Fields

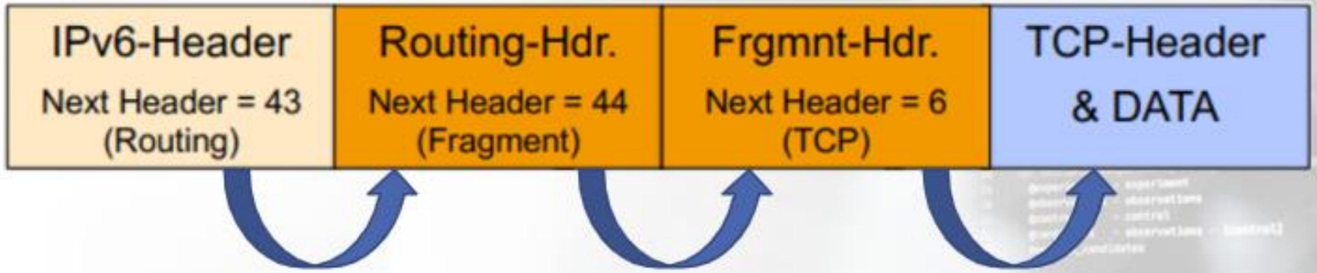
Note that Extension Headers are chained. For example:

The Next Header field is used so that the type of the protocol header that follows can be identified. More specifically, the Next Header field identifies the type of header that resides right after the IPv6 header, or the IPv6 Extension header that carries it.

Source: FIRST/SWITCH

1.3.3.1 Important IPv6 Fields

Note that Extension Headers are chained. Example:



Concerns:

- The number of EHs is not limited
- The number of options within an (Hop-by-Hop or Destination) Options Header is not limited
- There is no defined order of EHs (only a recommendation) (Exception: Hop-by-Hop Options Header must be first and non-recurring)
- EH have different formats

Threats:

1. High number of EHs could be used for FW/IDS/IPS/RA-Guard evasion
2. High number of EHs could be used to cause DoS to the destination
3. Manipulation/fuzzing of the EHs could be used to cause DoS to the destination
4. An attacker could use EHs for stealthy payload exchanges or covert communication

Source: FIRST/SWITCH

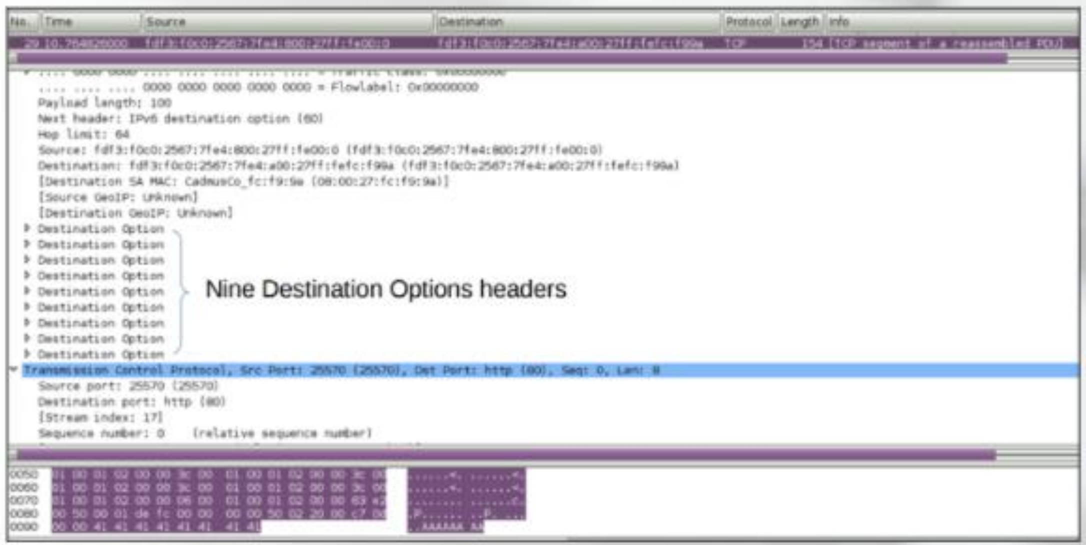
https://www.juniper.net/documentation/en_US/junos/topics/concept/port-security-ra-guard.html

OUTLINE

1.3.3.1 Important IPv6 Fields

Here we see an example of IDS evasion using a high number of EHs.

- In this case, 9 or more IPv6 Destination Options headers were sent in a single, unfragmented datagram for IDS evasion purposes.
- A variation of the attack could be 8 Dest Opt and 1 Frag Ext Hdr, or, 1 Hop-by-Hop, 1 Routing Header, 1 Dest Opt Header, 1 Fragment Header, 5 Dest Opt headers, etc.

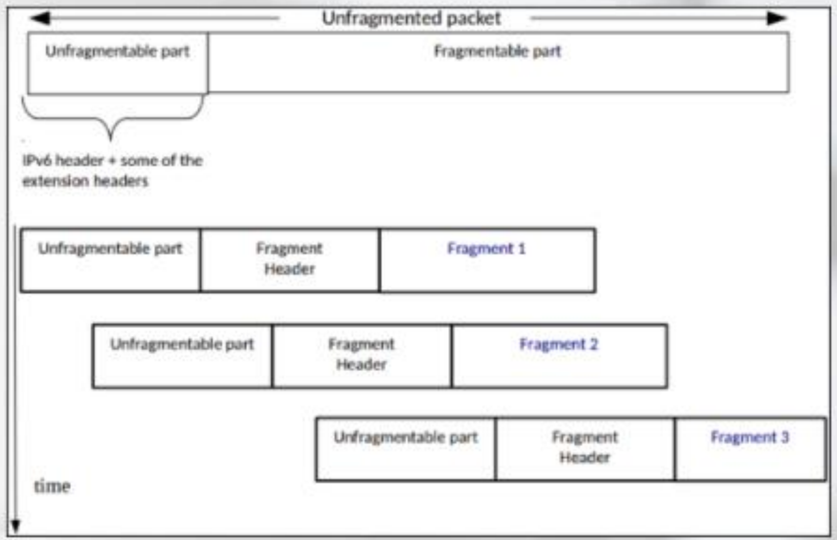


Source: Evasion of High-End IDPS Devices at the IPv6 Era

1.3.3.2 IPv6 Fragmentation

IPv6 Fragmentation

- The **Unfragmentable** Part will consist of the IPv6 main header plus any IPv6 Extension headers that must be processed by intermediate nodes en route to the destination (suppose that those are the Hop-by-Hop and the Routing IPv6 Extension headers in a later example).
- The **Fragmentable** Part will consist of the rest of the packet, that is, any IPv6 Extension headers that need be processed only by the final destination node(s), plus the upper-layer header and data (suppose that those are the Destination Options IPv6 Extension header, TCP header and its payload in a later example).
- Each fragment will be composed of:
    - The **Unfragmentable part** of the original packet, with the Payload Length of the original IPv6 header changed to contain the length of this fragment packet only (excluding the length of the IPv6 header itself), and the Next Header field of the last header of the Unfragmentable part changed to 44
    - A **Fragment header** containing the Next Header value that identifies the first header of the Fragmentable Part of the original packet.
      ... <snipped for brevity>...
      The fragment itself.



Source: Deering & Hinden, 1998 (RFC 2460)

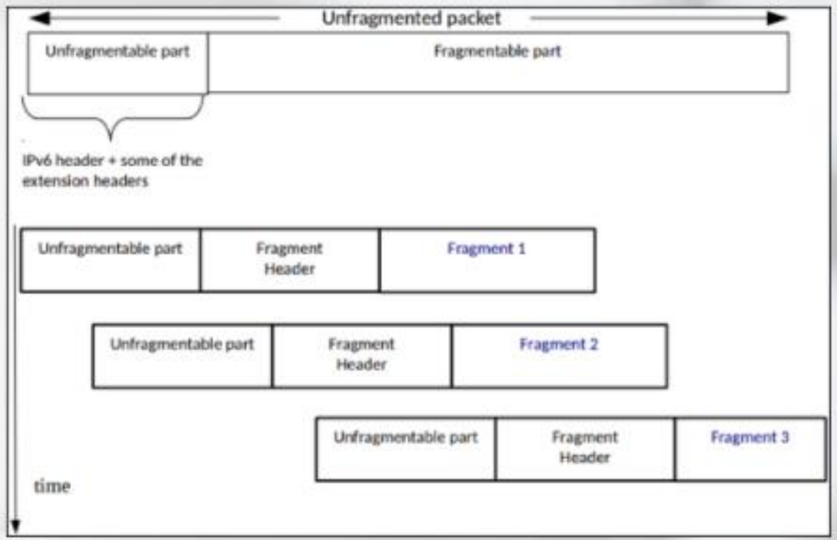
1.3.3.2 IPv6 Fragmentation

Reassembly of IPv6 fragmented diagrams

- "The Unfragmentable part of the reassembled packet consists of all headers up to, but not including, the Fragment header of the first fragment packet (that is, the packet whose Fragment Offset is zero), with the following change(s):
    - The Next Header field of the last header of the Unfragmentable Part is obtained from the Next Header field of the first fragment's Fragment header. ..."

- In addition, according to RFC 2460:

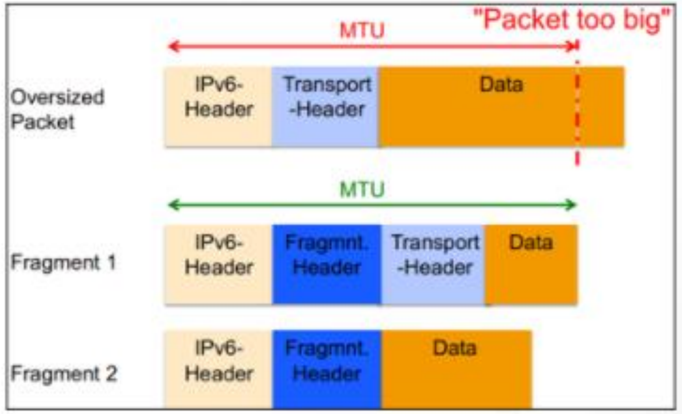
"The following conditions are not expected to occur, but are **not** considered errors if they do:
    - ... <snipped for brevity>...
    - The Next Header values in the Fragment headers of different fragments of the same original packet may differ. Only the value from the Offset zero fragment packet is used for reassembly. "



Source: Deering & Hinden, 1998 (RFC 2460)

1.3.3.2 IPv6 Fragmentation

Things will get worse with Fragmentation.



Threats:

1. DoS by sending a high number of incomplete fragment sets (M-flag 1)
2. IDS/IPS evasion by sending overlapping or nested fragments

Both the "fragmentable" and the "unfragmentable" parts may contain any IPv6 Extension headers.

Source: FIRST/SWITCH

OUTLINE

1.3.3.3 Abusing IPv6 Fragmentation & Detection

Let's analyze some real life examples of IDS/IPS evasion by combining incorrect usage of the Next Header values and legitimate fragmentation.

Example 1: Building legitimate but not expected fragmentation

What if an attacker crafted the following fragments?

- We notice that the Next Header value of the IPv6 Fragment Extension header is set to 6 (not 60 as in the first fragment and like it should be actually). Be careful not to mistake this as normal.
- Remember the "Reassembly of IPv6 fragmented diagrams" slide. Even in the case of such a crafted packet, the reassembly should occur by using only the Next Header value of the IPv6 Fragment Extension header whose offset is equal to 0

Fragment 1:

IPv6 header	IPv6 Routing Extension header	IPv6 Fragment Extension header	
Next Header Value = 43	Next Header Value = 44	Next Header Value = 60	(part 1 out of 2 of the fragmentable part)

Fragment 2:

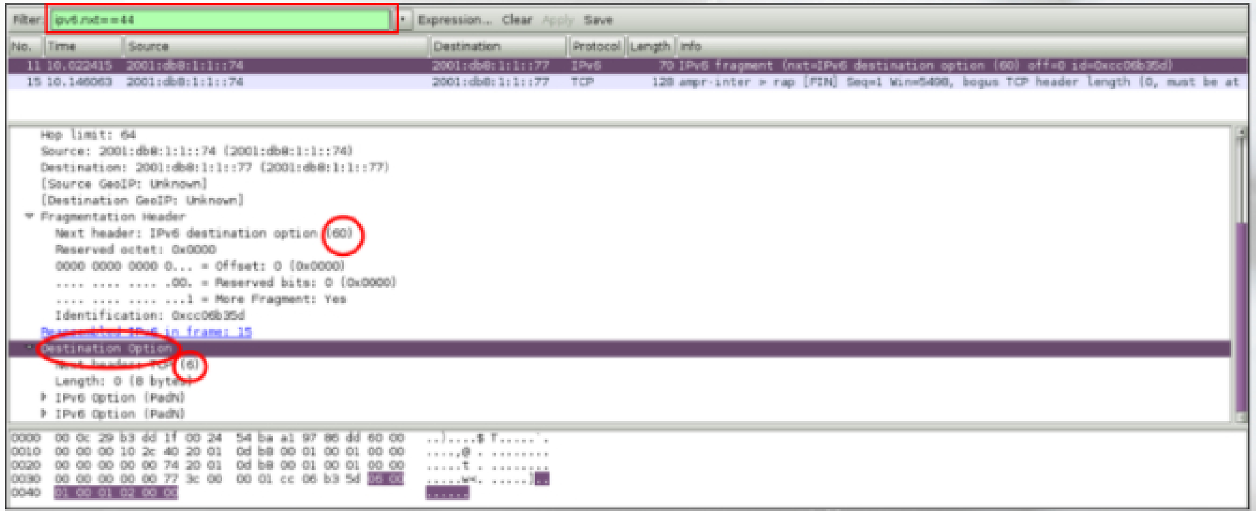
IPv6 header	IPv6 Routing Extension header	IPv6 Fragment Extension header	
Next Header Value = 43	Next Header Value = 44	Next Header Value = 6	(part 2 out of 2 of the fragmentable part)

Suppose that the upcoming Layer 4 protocol is TCP

Source: Evasion of High-End IDPS Devices at the IPv6 Era

Example 1: First fragment of the attack

- An IPv6 Destination Options Extension Header follows the Fragmentation Header; this is due to the Next Header value of the IPv6 Fragment Extension header being 60.
- The Layer 4 header is TCP. This is due to the Next Header value of the IPv6 Destination Options header being 6.
- Please, observe that the bytes of the IPv6 Destination Options header are as following (highlighted): 06 00 01 00 01 02 00 00.



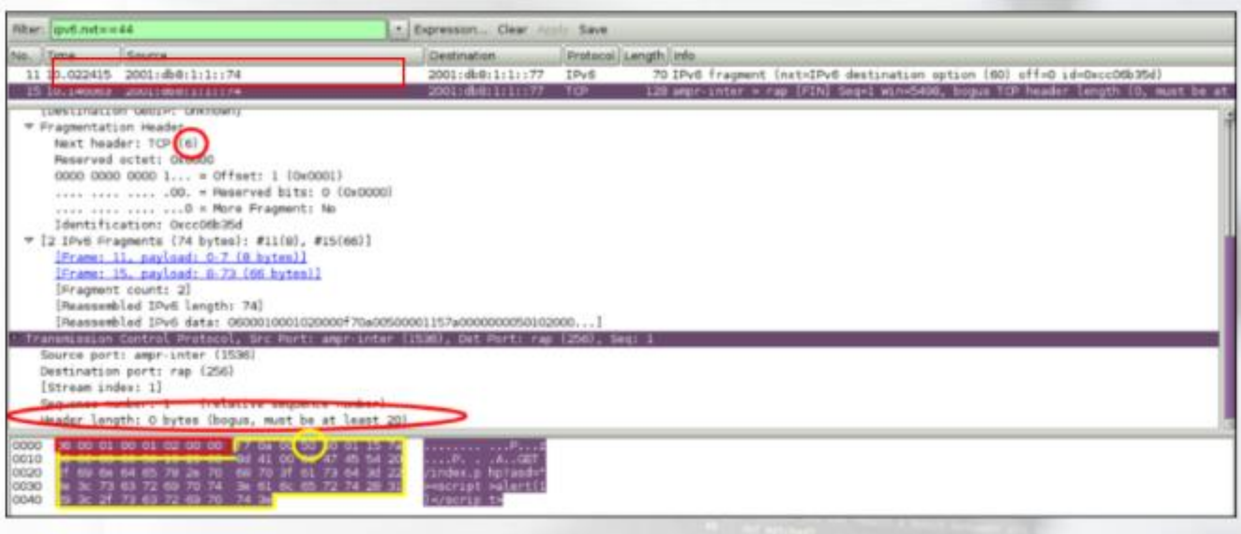
Source: Evasion of High-End IDPS Devices at the IPv6 Era

1.3.3.3 Abusing IPv6 Fragmentation & Detection

Example 1: The reassembled datagram

- The Next Header value of the IPv6 Fragment Extension header is 6 (not 60 like we saw on the first fragment). In purple, is the Layer 4 header (TCP) and its payload. Wireshark also indicates a destination port of 256.
- Next, Wireshark warns us of a bogus Header length. Specifically, the Header length seems to be 0, while it should be at least 20.
- If we look at the first bytes of the TCP header, we notice the following sequence : 06 00 01 00 01 02 00 00. If you recall, that's the IPv6 Destination Options Extension header carried by the first fragment.
- What happened is that due to the Next Header value of the IPv6 Fragment Extension Header of the second fragment being 6, Wireshark misinterpreted the fragmentable part and analyzed the IPv6 Destination Options Extension header (red rectangle) as part of the TCP header (purple, highlighted text). This is how the Destination Port 256 came up.

A Tipping Point IDPS version back then, suffered from the same kind of misinterpretation, resulting in don't "seeing" those fragmented packets as HTTP traffic and allowing them to reach and exploit the destination.



Source: Evasion of High-End IDPS Devices at the IPv6 Era

OUTLINE

1.3.4 IPv6 Tunneling

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

1.3.4 IPv6 Tunneling

We will talk about flows in an upcoming module, but the thing is that you can detect IPv6 tunnels inside network logs or NetFlow records.

For example:

- IPv4 Protocol type 41 (ISATAP, 6to4 traffic)
- IPv4 to UDP 3544 (Teredo traffic)
- Traffic to 192.88.99.1 (6to4 anycast server)
- DNS server log: resolution of "ISATAP"

https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/enterprise-ipv6-solution/white_paper_c11-629391.pdf

1.3.4 IPv6 Tunneling

IPv6 tunnel detection with Wireshark (and a nice IPv6 analysis primer in general) can be found on the following resource:

https://sharkfestus.wireshark.org/sharkfest.11/presentations/B-3_Leutert-Discovering_IPv6_with_Wireshark.pdf

1.3.4 IPv6 Tunneling

Consider the *ipv6_tunnel_packets.pcap file* found in this module's resources. Can you identify the security issue?

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.73.148	64.233.169.104	TCP	74	42419 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=4…
2	0.029029	64.233.169.104	192.168.73.148	TCP	58	80 → 42419 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	0.043644	192.168.73.148	64.233.169.104	TCP	54	42419 → 80 [ACK] Seq=1 Ack=1 Win=5840 Len=0
4	8.850179	192.168.73.148	63.245.209.93	TCP	54	40805 → 80 [FIN, ACK] Seq=1 Ack=1 Win=7010 Len=0
5	8.850236	63.245.209.93	192.168.73.148	TCP	54	80 → 40805 [ACK] Seq=1 Ack=2 Win=64239 Len=0
6	9.829544	fe80::ffff:ffff:ffff	ff02::2	ICMPv6	103	Router Solicitation
7	9.956850	fe80:0:7274:696e:8000:dd…	fe80::ffff:ffff:ffff	ICMPv6	159	Router Advertisement

1.3.4 IPv6 Tunneling

Consider the *ipv6_tunnel_packets.pcap file* found in this module's resources. Can you identify the security issue?

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.73.148	64.233.169.104	TCP	74	42419 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=4…
2	0.029029	64.233.169.104	192.168.73.148	TCP	58	80 → 42419 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	0.043644	192.168.73.148	64.233.169.104	TCP	54	42419 → 80 [ACK] Seq=1 Ack=1 Win=5840 Len=0
4	8.850179	192.168.73.148	63.245.209.93	TCP	54	40805 → 80 [FIN, ACK] Seq=1 Ack=1 Win=7010 Len=0
5	8.850236	63.245.209.93	192.168.73.148	TCP	54	80 → 40805 [ACK] Seq=1 Ack=2 Win=64239 Len=0
6	9.829544	fe80::ffff:ffff:ffff	ff02::2	ICMPv6	103	Router Solicitation
7	9.956850	fe80:0:7274:696e:8000:dd…	fe80::ffff:ffff:ffff	ICMPv6	159	Router Advertisement

- Frames 1 to 5 show the connection attempts to 64.233.169.104 (google.com) being closed [Notice the FIN, ACK on frame 4]

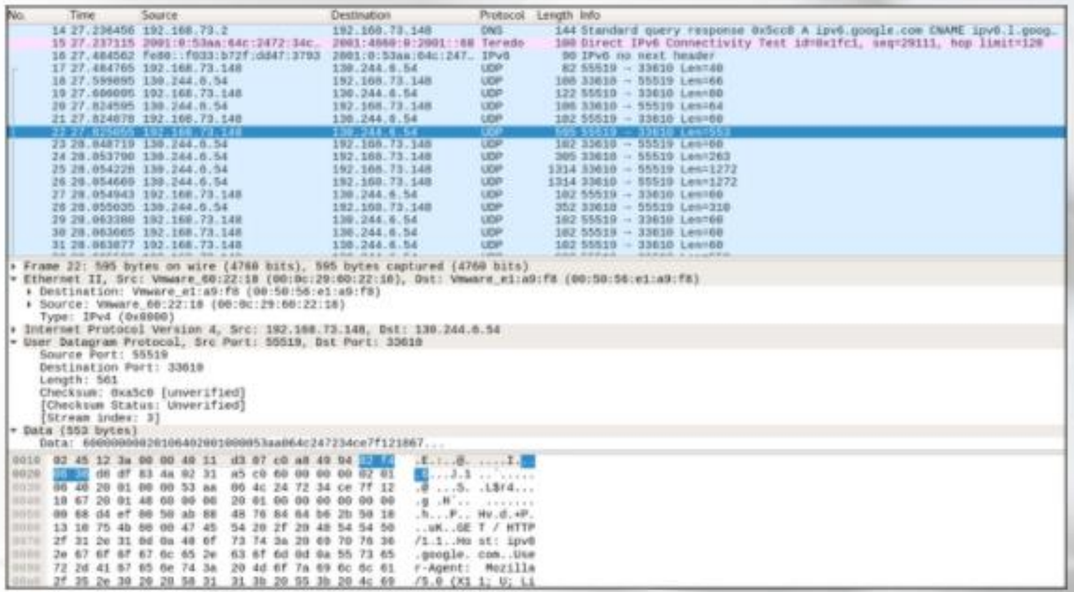
- Frame 6 and 7 are actually IPv6 packets being transmitted by IPv4 UDP. They actually depict a Router Solicitation and a Router Advertisement. We will analyze both in just a bit. For now, they can be seen as the IPv6 mechanism to ask for an IPv6 address and offer an IPv6 prefix.

1.3.4 IPv6 Tunneling

Consider the *ipv6_tunnel_packets.pcap file* found in this module's resources. Can you identify the security issue?

- If we now carefully look at frame 22, we see an HTTP request being made to ipv6.google.com. Specifically, the HTTP GET string is being carried by a UDP packet.
- If we right-click on this frame, then **Follow** and finally **UDP Stream**, we will also see the request was successful.

This is most probably a case of IPv6 tunneling protocols being used to bypass an IPv4-only firewall.

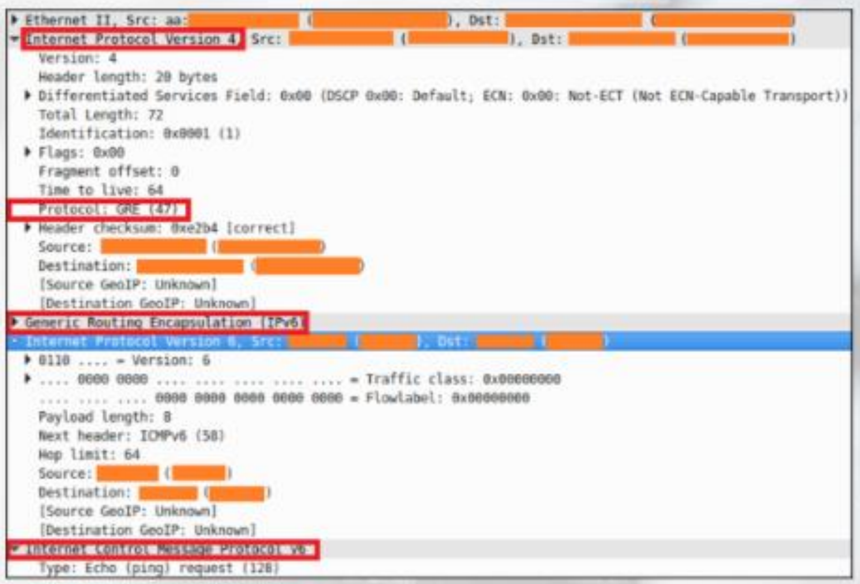


OUTLINE

1.3.4 IPv6 Tunneling

Finally, IPv6 packets over IPv4 can be transmitted inside a Generic Routing Encapsulation (GRE) tunnel.

For this to happen, tunnel software-assisted encapsulation and de-encapsulation is required.



1.3.5 ICMPv6

ICMPv6 is much more complex than ICMP. Below is a short overview before we dive deeper into it and its usages. It inherits and extends the functionalities of ICMP version 4.

Error-messages (1-127) 1:Destination Unreachable, 2:Packet too big (PMTUD), 3:Time Exceeded (Hop Limit), 4:Parameter Problem
Info-Messages (Ping) 128:Echo Request, 129:Echo Reply
Multicast Listener Discovery (MLD, MLD2) 130:Multicast Listener Query, 131/143:Multicast Listener Report/2 132:Multicast Listener Done
Neighbor Discovery (NDP), Stateless Autoconfiguration (SLAAC) 133:Router Solicitation, 134:Router Advertisement, 135:Neighbor Solicitation (DAD), 136:Neighbor Advertisement (DAD), 137:Redirect Message
Other (Router Renumbering, Mobile IPv6, Inverse NS/NA,...) 138-153

OUTLINE

ICMPv6 Usages

- IPv6's ICMPv6 Neighbor Discovery Protocol (NDP) is the equivalent of ARP and ICMP router discovery and redirect in IPv4.
- IP/MAC association in IPv6 is conducted through Neighbor Solicitation (NS) (ICMPv6 type 135) and Neighbor Advertisement (NA) (ICMPv6 type 136) messages.
- The default gateway is identified via Router Solicitation (RS) (ICMPv6 type 133) and Router Advertisement (RA) (ICMPv6 type 134) messages

OUTLINE

1.3.5 ICMPv6

In IPv6, the concept of Stateless Autoconfiguration (SLAAC) exists.

- Stateless autoconfiguration allows a node to be configured without any configuration server.
- Interaction between a node and a local IPv6 router is required for the node to configure its own globally routable addresses.
- The address combines the adapter MAC address with network prefixes identified through interaction with the neighboring router.
- Multihomed hosts perform autoconfiguration for each interface.
- Stateless autoconfiguration uses the Neighbor Discovery protocol.

OUTLINE

1.3.5 ICMPv6

SLAAC

Stateless Autoconfiguration steps

- Link-Local Address Generation: The device generates a link-local address.
- Ensure Link-Local Address is Unique:
  - Is there a host with the same address?
  - A Neighbor Solicitation message is sent
  - Listens for a Neighbor Advertisement
- Link-Local Address Assignment: Address used for local network communication only
- Router Contact:
  - Consult with the local router
  - A Router Solicitation message is sent
  - Listen for a Router Advertisement
- Router Direction:
  - Stateful or Stateless?
  - Prefix?
- Global Address Configuration: Global unicast address is formed by combining the MAC address (IID) and the network prefix

OUTLINE

1.3.5 ICMPv6

Let's now see how those important ICMPv6 messages look like on the wire.

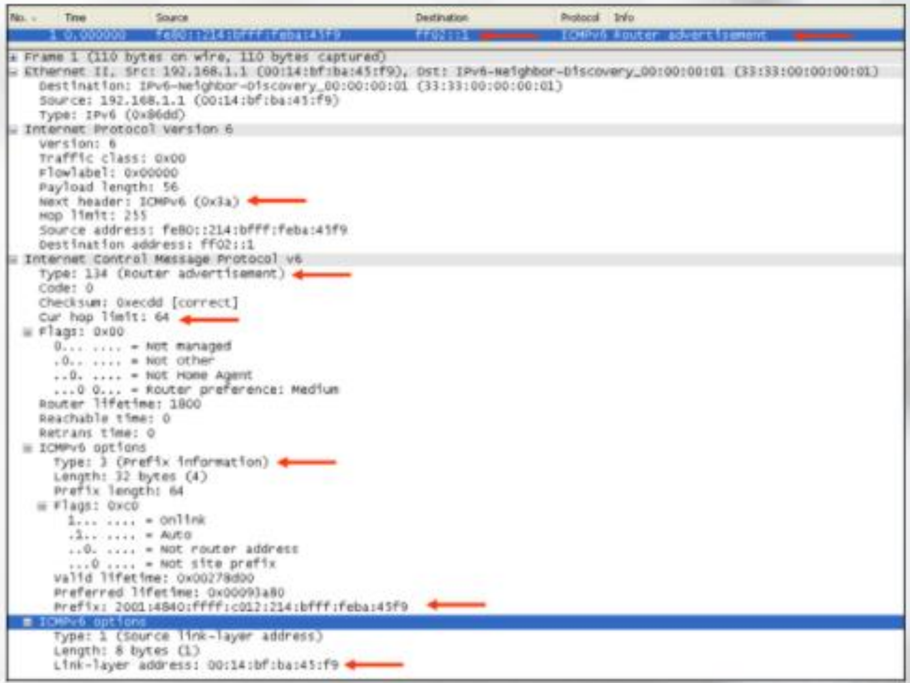
OUTLINE

1.3.5 ICMPv6

Router Advertisement Packet

Source Address: Must be the link-local addresses assigned to the interface from which this message is sent.

Destination Address: Usually the Source Address from where a Router Solicitation originated or the all-nodes multicast address



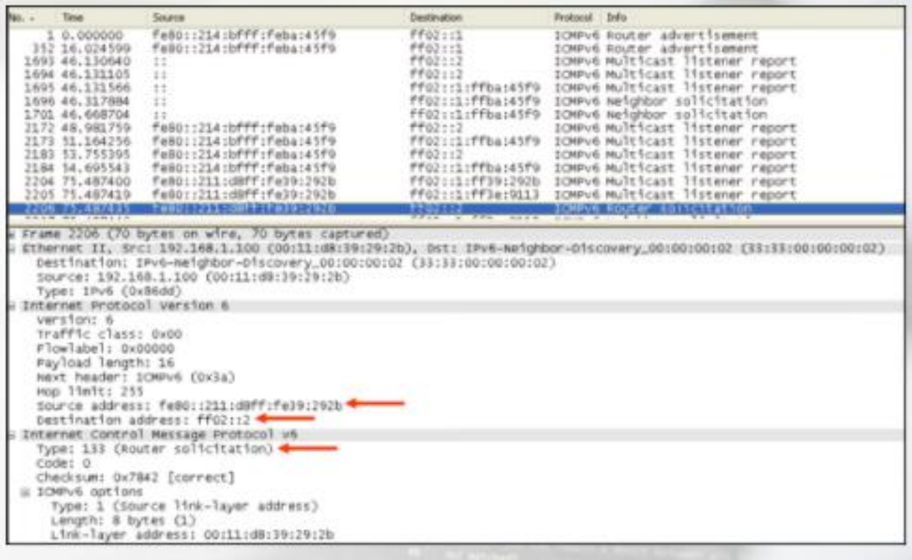
OUTLINE

1.3.5 ICMPv6

Router Solicitation Packet

Source Address: Usually 0:0:0:0:0:0:0:0 or the configured unicast address of the interface.

Destination Address: Usually the all-routers multicast address (FF02::2).



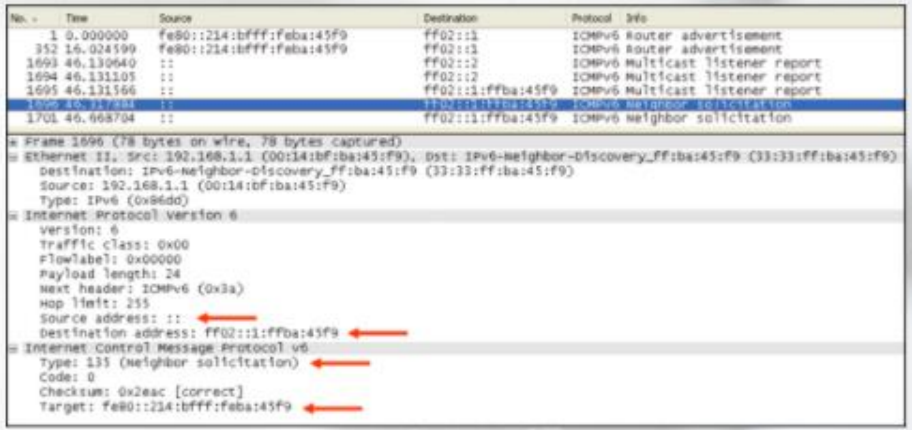
OUTLINE

1.3.5 ICMPv6

Neighbor Solicitation Packet

Source Address: An addresses assigned to the interface from which this message is sent or 0:0:0:0:0:0:0:0.

Destination Address: The solicited-node multicast address corresponding to the target address or the target address.

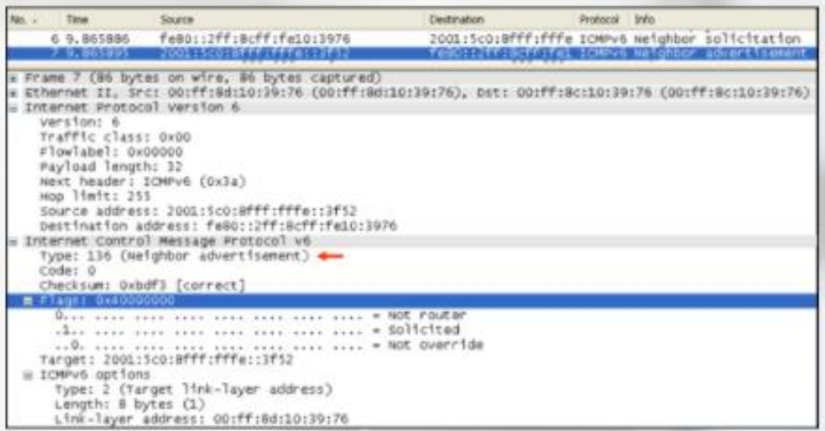


OUTLINE

1.3.5 ICMPv6

Neighbor Advertisement

A node sends Neighbor Advertisements in response to Neighbor Solicitations and sends unsolicited Neighbor Advertisements in order to (unreliably) propagate new information quickly.

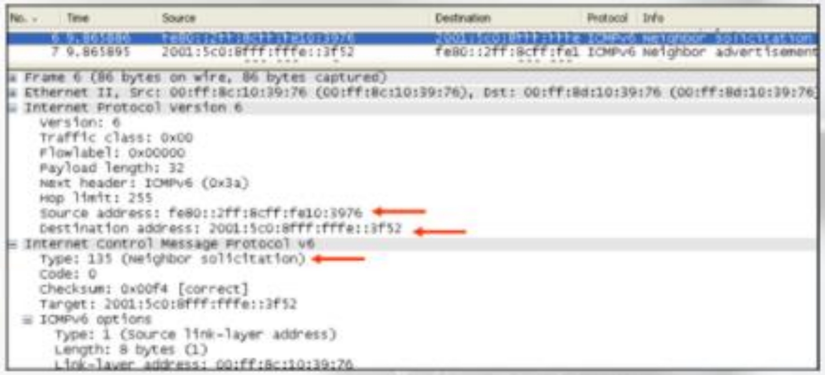


OUTLINE

1.3.5 ICMPv6

Neighbor Solicitation Packet

To a specific unicast address
Duplicate Address Detection.



1.3.6 IPv6 Security Shortcomings

Let's now analyze some of IPv6's security shortcomings.

Since we just analyzed IPv6's Neighbor Discovery Protocol, let's focus on some of its weaknesses.

1.3.6 IPv6 Security Shortcomings

If you recall, when IPv4 is in place, an attacker can misuse ARP and ICMPv4 messages to manipulate traffic.

Unfortunately, this is the case with IPv6 as well; this time NDP messages can be misused to achieve network traffic manipulation.

1.3.6 IPv6 Security Shortcomings

Network Discovery Attacks

Attackers ultimately want to introduce incorrect IPv6 host address/link layer pairings; this can be achieved via two (2) distinct ways:

- 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.

- An attacker can also craft an NS request containing the fake IPv6 host address/link layer pairing. Listening neighbors will introduce this ill-intended pairing in their neighbor cache; this is the equivalent of abused Gratuitous ARP in IPv4.

1.3.6 IPv6 Security Shortcomings

Network Discovery Attacks

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.

OUTLINE

1.3.6 IPv6 Security Shortcomings

Network Discovery Attacks

What you should also be aware of is that Secure Neighbor Discovery (SEND) exists, ensuring message integrity and enforcing message source/sender association. It does so by utilizing a timestamp and a nonce.

More information about SEND can be found in RFC 3971.

https://tools.ietf.org/html/rfc3971

1.3.6 IPv6 Security Shortcomings

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

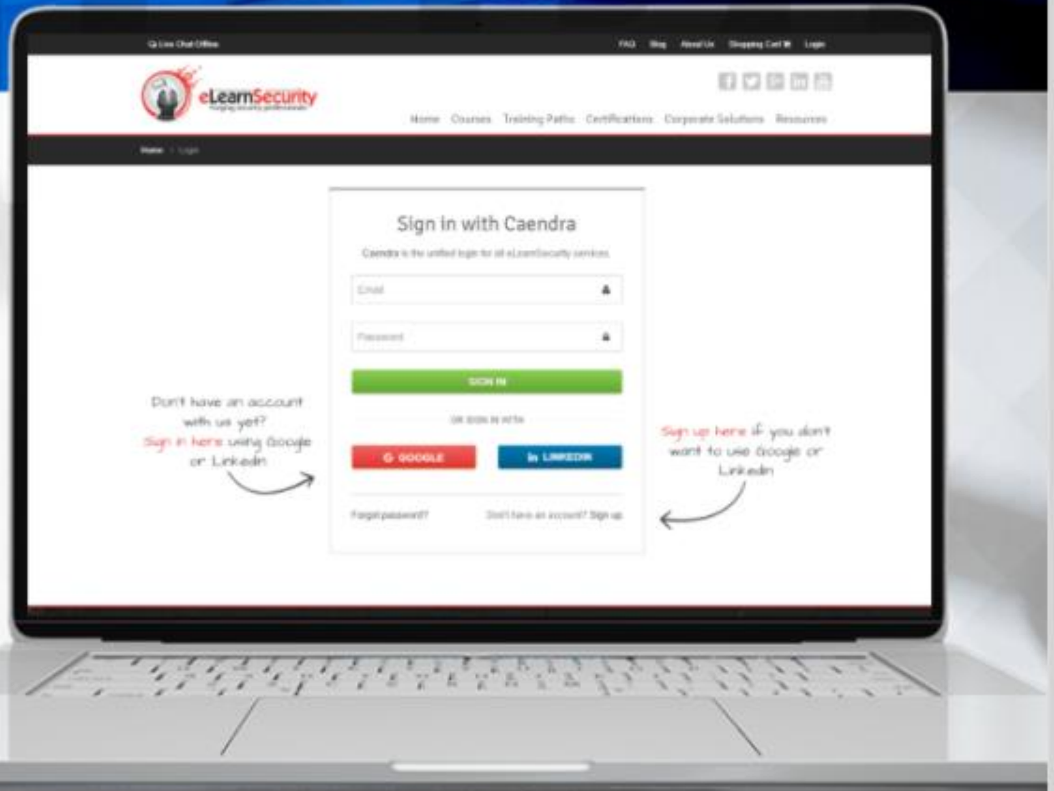
3. https://www.tno.nl/media/3274/testing_the_security_of_ipv6_implementations.pdf

Lab 1 for Intrusion Detection by Analyzing Traffic

Traffic Analysis Challenges

During this lab you will:

- Refresh your networking knowledge
- Learn to identify TCP spoofing and internal botnet-like activity
- Practice identifying attacks by analyzing network traffic, including IPv6-based ones



**To access, go to the course in your members area and click the resources drop-down in the appropriate module line to access the files for this offline lab.*

References

OUTLINE

References

Wireshark

https://www.wireshark.org/

MAC Address Lookup

https://www.macvendorlookup.com/

RFC 826

https://www.ietf.org/rfc/rfc826.txt

Wireshark Display Filter Reference for ARP

https://www.wireshark.org/docs/dfref/a/arp.html

References

Colasoft's Capsa

https://www.colasoft.com/download/products/download_capsa.php

RFC 791

https://www.ietf.org/rfc/rfc791.txt

IANA IPv4 Address Space Registry

https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml

CVE-2018-5391

https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/ADV180022

References

GestióIP IPv6 resources

http://www.gestioip.net/docu/ipv6_address_examples.html

RFC 2460

https://tools.ietf.org/html/rfc2460#page-4

Internet Protocol Version 6 Address Space

https://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml

RA-Guard

https://www.juniper.net/documentation/en_US/junos/topics/concept/port-security-ra-guard.html

References

NetFlow records

https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/enterprise-ipv6-solution/white_paper_c11-629391.pdf

Discovering IPv6 with Wireshark

https://sharkfestus.wireshark.org/sharkfest.11/presentations/B-3_Leutert-Discovering_IPv6_with_Wireshark.pdf

RFC 3971

https://tools.ietf.org/html/rfc3971

Ripe NCC – IPv6 Security

https://www.ripe.net/support/training/material/ipv6-security/ipv6security-slides.pdf

References

IPv6 Attack & Defense Strategies

https://www.blackhat.com/docs/sp-14/materials/arsenal/sp-14-Schaefer-Workshop-Slides.pdf

Testing the Security of IPv6 Implementations

https://www.tno.nl/media/3274/testing_the_security_of_ipv6_implementations.pdf

Labs

Traffic Analysis Challenges

During this lab you will:

- Refresh your networking knowledge
- Learn to identify TCP spoofing and internal botnet-like activity
- Practice identifying attacks by analyzing network traffic, including IPv6-based ones

To access, go to the course in your members area and click the resources drop-down in the appropriate module line to access the files for this offline lab.