

v4

Penetration Testing Student

Networking

Section 01 | Module 02

```
1 def initialize(experiment, observations = [], control = nil)
2   @experiment = experiment
3   @observations = observations
4   @control = control
5   @candidates = []
6   evaluate_candidates
7 end
8
9 # Public: the experiment's context
10 def context
11   experiment.context
12 end
13
14 # Public: the name of the experiment
15 def experiment_name
16   experiment.name
17 end
18
19 # Public: was the result a match between the
20 def matched?
21   lib/scientist/result.rb 1

```

© Caendra Inc. 2019
All Rights Reserved

Table of Contents

Module 02 | Networking

2.1 Protocols

2.2 Internet Protocol (IP)

2.3 Routing

2.4 Link Layer Devices and
Protocols

2.5 TCP and UDP

2.6 Firewalls and Network
Defense

2.7 DNS

2.8 Wireshark



Learning Objectives

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

- Modern network protocols
- Ways computers talk to each other
- Sniffing and capturing the network traffic



Protocols



2.1 Protocols

How does this support my pentesting career?

- Protocols are used in every computer network communication
- You need to know how things work to exploit them

2.1 Protocols

In a computer network, machines talk to each other by means of **protocols**.

These protocols ensure that different computers, using different hardware and software, can communicate.



2.1 Protocols

There is a large variety of networking protocols on the Internet, each one with its own purpose.

We are going to discuss a few of them in detail and point you towards free online resources for others.



2.1.1 Packets

The primary goal of networking is to exchange information between networked computers; this information is carried by **packets**.

Packets are nothing but streams of bits running as electric signals on physical media used for data transmission. Such media can be a **wire** in a LAN or **the air** in a WiFi network.

These electrical signals are then interpreted as bits (zeros and ones) that make up the information.



2.1.1 Packets

Every packet in every protocol has the following structure.

Header

Payload

2.1.1 Packets

The **header** has a protocol-specific structure: this ensures that the receiving host can correctly interpret the payload and handle the overall communication.



Header

Payload



2.1.1 Packets

The **payload** is the actual information. It could be something like part of an email message or the content of a file during a download.



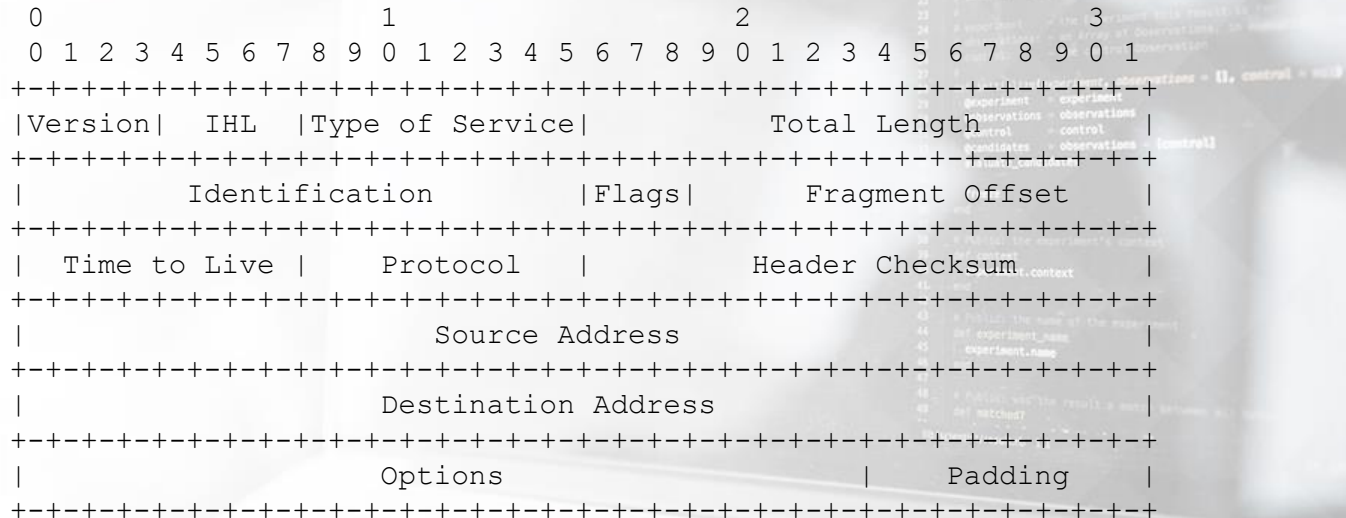
Header

Payload



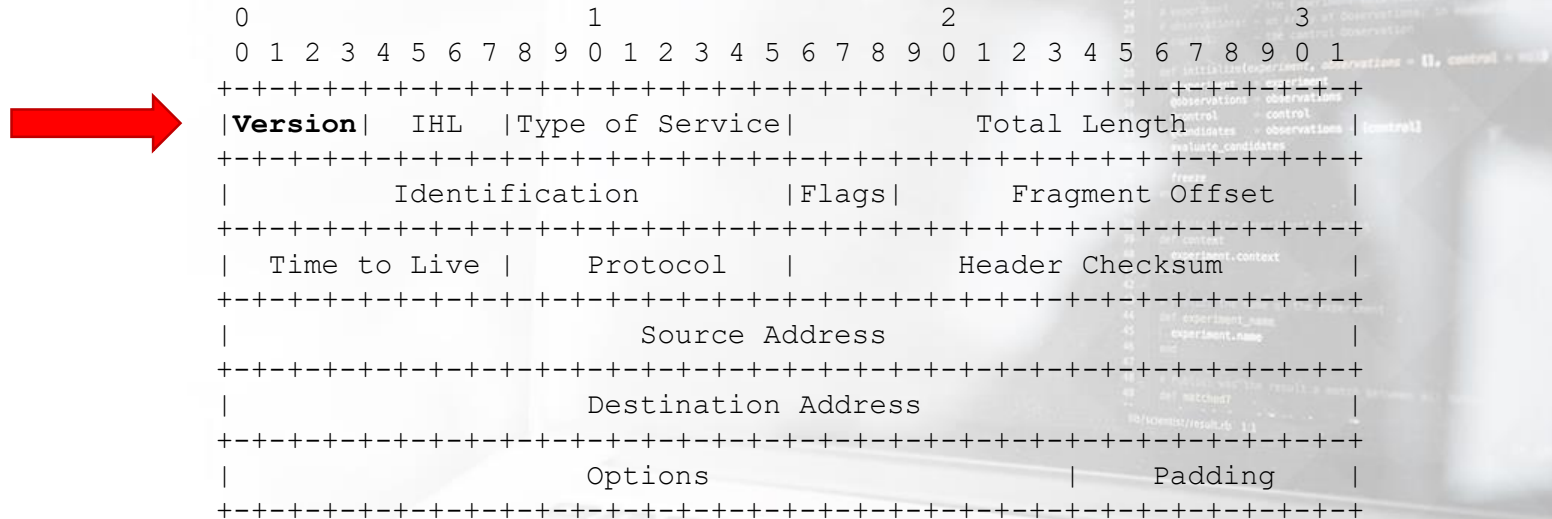
2.1.1.1 Example – The IP Header

For example, the IP protocol header is at least 160 bits (20 bytes) long, and it includes information to interpret the content of the IP packet.



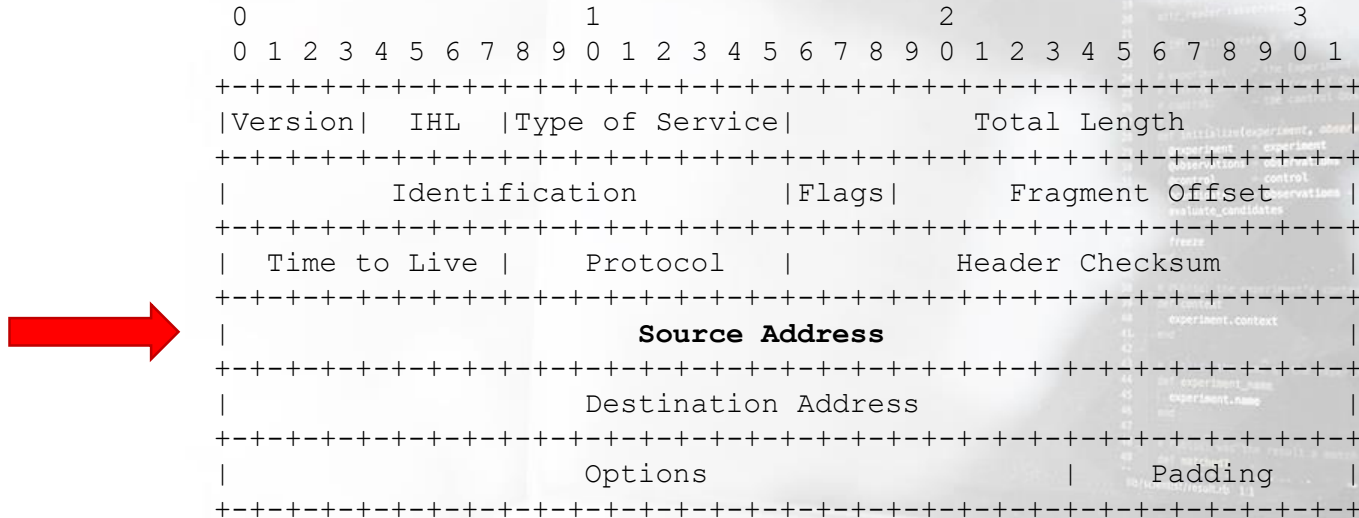
2.1.1.1 Example – The IP Header

The first four bits identify the **IP version**. Today they can be used to represent IP version 4 or 6.




2.1.1.1 Example – The IP Header

The 32 bits starting at position 96 represent the **source address**.



2.1.1.1 Example – The IP Header

The following four bytes represent the **destination address**.



```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|                    Total Length|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Identification                      |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

```


2.1.1.1 Example – The IP Header

Using the information in the header, the nodes involved in the communication can understand and use IP packets.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|                    Total Length|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Identification                    |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |           Header Checksum           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

```

2.1.2 Protocol Layers

In the previous example, we saw the header of the IP (**Internet Protocol**). There are many protocols out there, each one for a specific purpose. Purposes such as:

- Exchanging emails, files or performing VoIP calls
- Establishing a communication between a server and a client
- Identifying computers on a network
- Transmitting data



2.1.2 Protocol Layers

Instead of using specific examples, let's rewrite the previous list focusing on the features that a protocol provides:

- Make an application (email client, FTP, browser, ...) work
- Transport data between processes (the server and the client programs)
- Identify hosts
- Use the physical media to send packets



2.1.2 Protocol Layers

Moreover, we can rewrite the list again as:

- Application layer
- Transport layer
- Network layer
- Physical layer



These **layers** work on top of one another, and every layer has its own **protocol**.

```
21 # @param: Create a new experiment
22 # @param: experiment_name: the experiment's name
23 # @param: observations: an array of Observations, in which
24 # @param: control: the control Observation
25 # @param: candidates: the candidates to be evaluated
26
27 def initialize(experiment, observations = [], control = null)
28   @experiment = experiment
29   @observations = observations
30   @control = control
31   @candidates = observations - [control]
32   evaluate_candidates
33
34   freeze
35 end
36
37 # @param: the experiment's context
38 def context
39   @context
40 end
41
42 # @param: the name of the experiment
43 def experiment_name
44   @experiment_name
45 end
46
47 def mechasy
48   @mechasy
49 end
50
51 # @param: result: the result of the experiment
52 def result
53   @result
54 end
```



2.1.2 Protocol Layers

Each layer serves the one above it.

The **application layer** does not need to know how to **identify a process** on a host, how to **reach it** and how to **use the copper wire** to establish a communication.

It just uses its underlying layers.



2.1.3 ISO/OSI

In 1984, the International Organization for Standardization (ISO) published a theoretical model for network systems communication: the Open System Interconnection (OSI) model.

The **ISO/OSI** model was never implemented, but it is widely used in literature or when talking about IT networks.



2.1.3 ISO/OSI

ISO/OSI consists of seven layers and is used as a reference for the implementation of actual protocols.

You can find more information about ISO/OSI [here](#).

Application

Presentation

Session

Transport

Network

Data Link

Physical

2.1.4 Encapsulation

But how do protocols work together? If every protocol has a header and a payload, how can a protocol use the one on its lower layer?

The idea is simple. The **entire upper protocol packet** (header plus payload) is the **payload** of the lower one; this is called **encapsulation**.

```
21 def _create_experiment(self, experiment_name, observations, control):
22     """Create a new experiment.
23     """
24     # The experiment will result in a new
25     # observations - an array of Observations, an array
26     # control - the control observation
27
28     # Create the experiment
29     @experiment = experiment
30     @observations = observations
31     @control = control
32     @candidates = observations - [control]
33     evaluate_candidates
34
35     # Create the experiment's context
36     context = Context()
37     context.set_experiment_name(experiment_name)
38     context.set_observations(observations)
39     context.set_control(control)
40     context.set_candidates(candidates)
41     context.set_result(result)
42     return context
43
44 def experiment_name(self):
45     """Return the name of the experiment.
46     """
47     return self._experiment_name
48
49 def _match(self, result):
50     """Return the result a match between the
51     """
52     return result == self._result
53
54 def _result(self):
55     """Return the result of the experiment.
56     """
57     return self._result
58
59 def _result(self):
60     """Return the result of the experiment.
61     """
62     return self._result
```



2.1.4 Encapsulation

In the following slides, you will see how encapsulation is used by the IP protocol suite, or TCP/IP.

TCP/IP is a real-world implementation of a networking stack and is the protocol stack used on the Internet.

```
21 # @param: control = the control observation
22 # @param: observations = an array of Observations, in which the control observation is included
23 # @param: control = the control observation
24 # @return: the result of the experiment
25
26 def initialize(experiment, observations = [], control = null)
27   @experiment = experiment
28   @observations = observations
29   @control = control
30   @candidates = observations - [control]
31   evaluate_candidates
32 end
33
34 # @param: the experiment's control
35 # @return: the result of the experiment
36
37 def experiment_name
38   @experiment_name
39 end
40
41 # @param: the result of the experiment
42 # @return: the result of the experiment
43
44 def experiment_name
45   @experiment_name
46 end
47
48 # @param: the result of the experiment
49 # @return: the result of the experiment
50
51 def match
52   @match
53 end
54
55 # @param: the result of the experiment
56 # @return: the result of the experiment
57
58 def result
59   @result
60 end
```



2.1.4 Encapsulation

TCP/IP has four layers:

Application

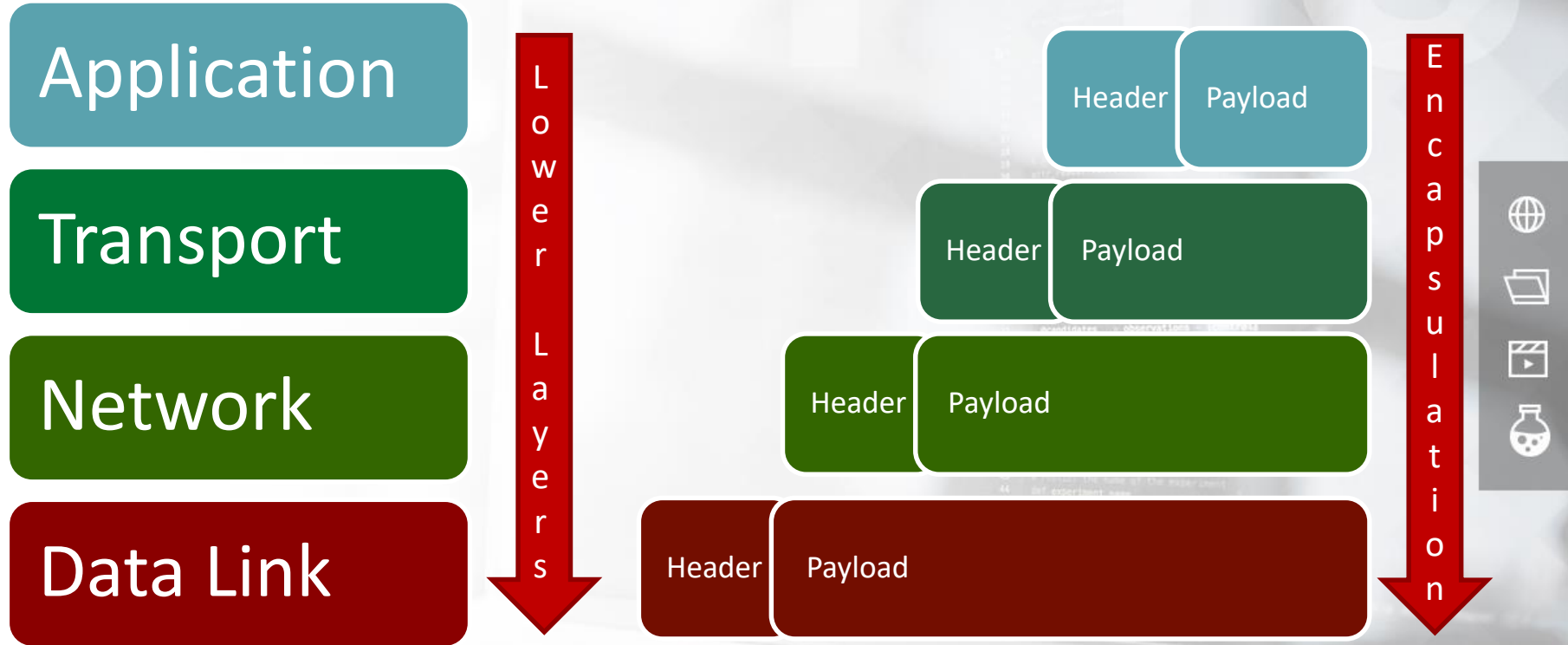
Transport

Network

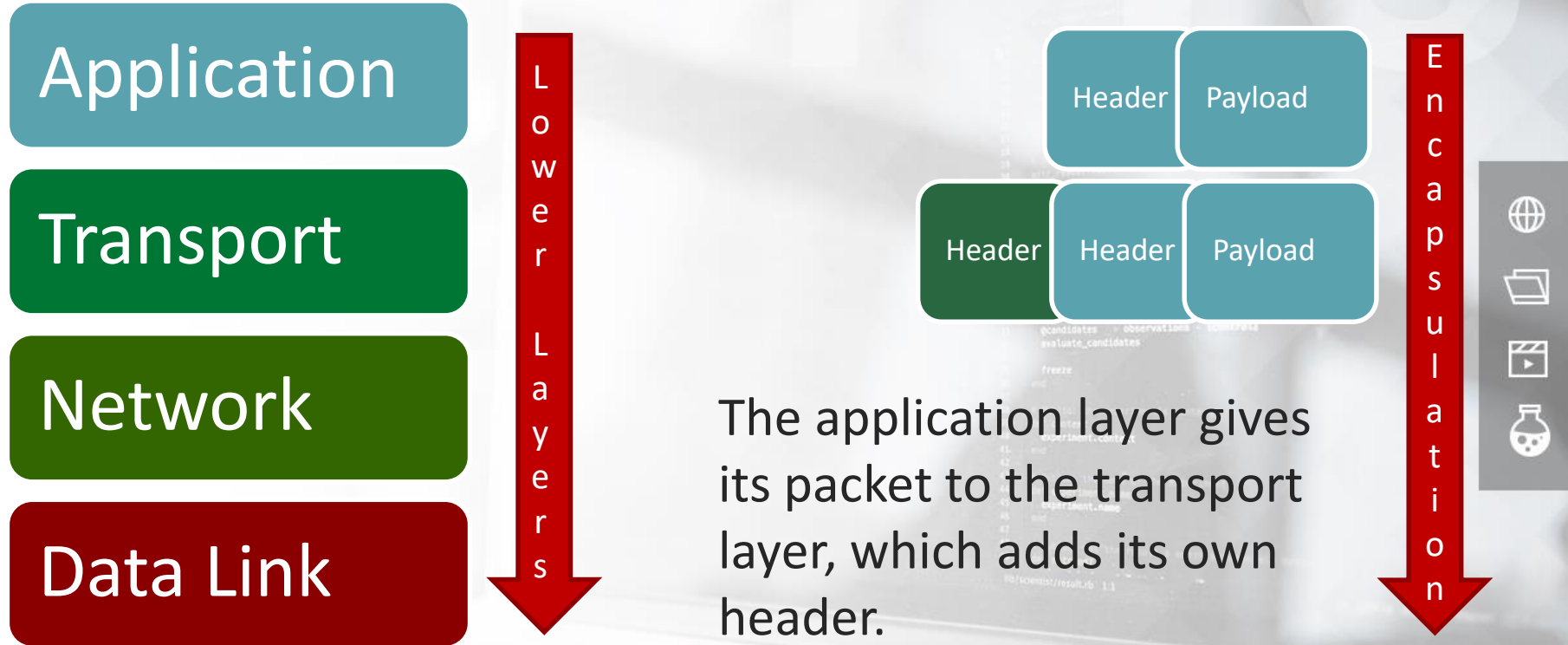
Data Link

You will learn how TCP/IP works in the remainder of this module.

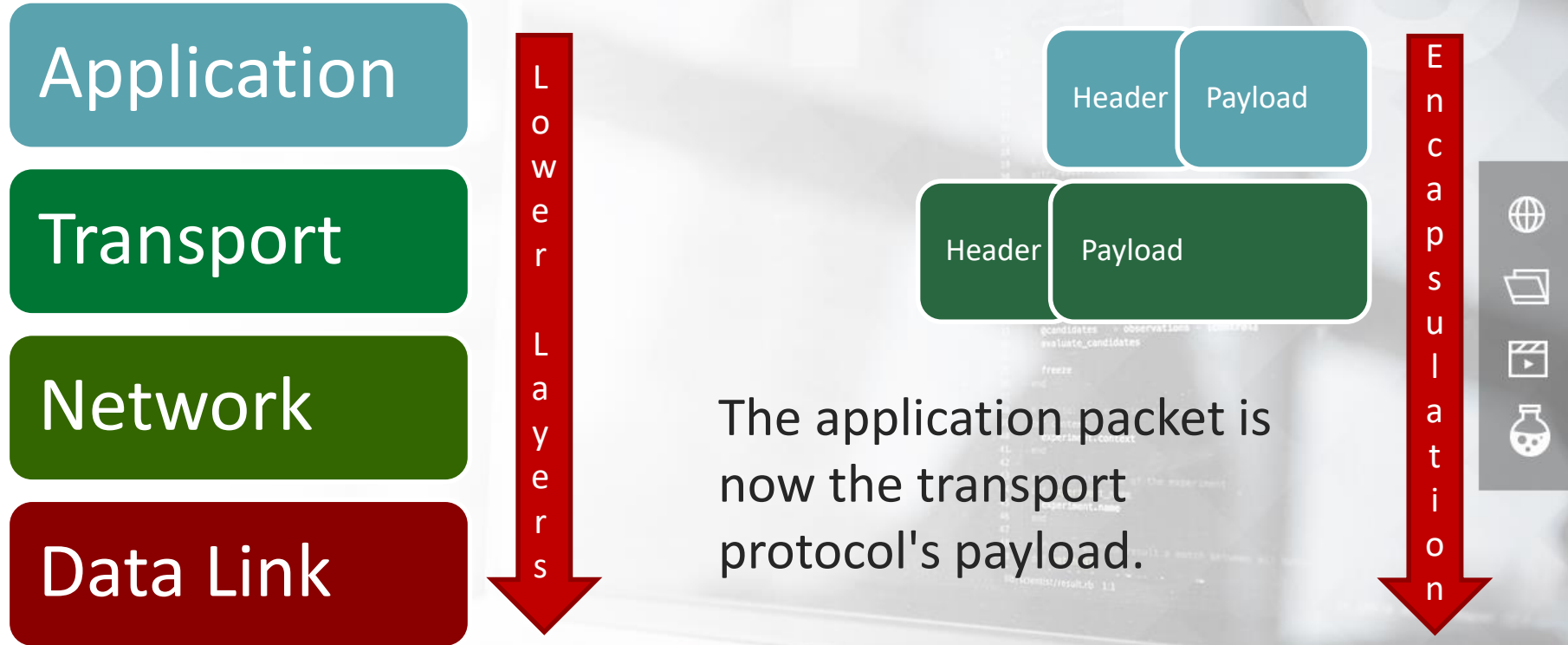
2.1.4 Encapsulation



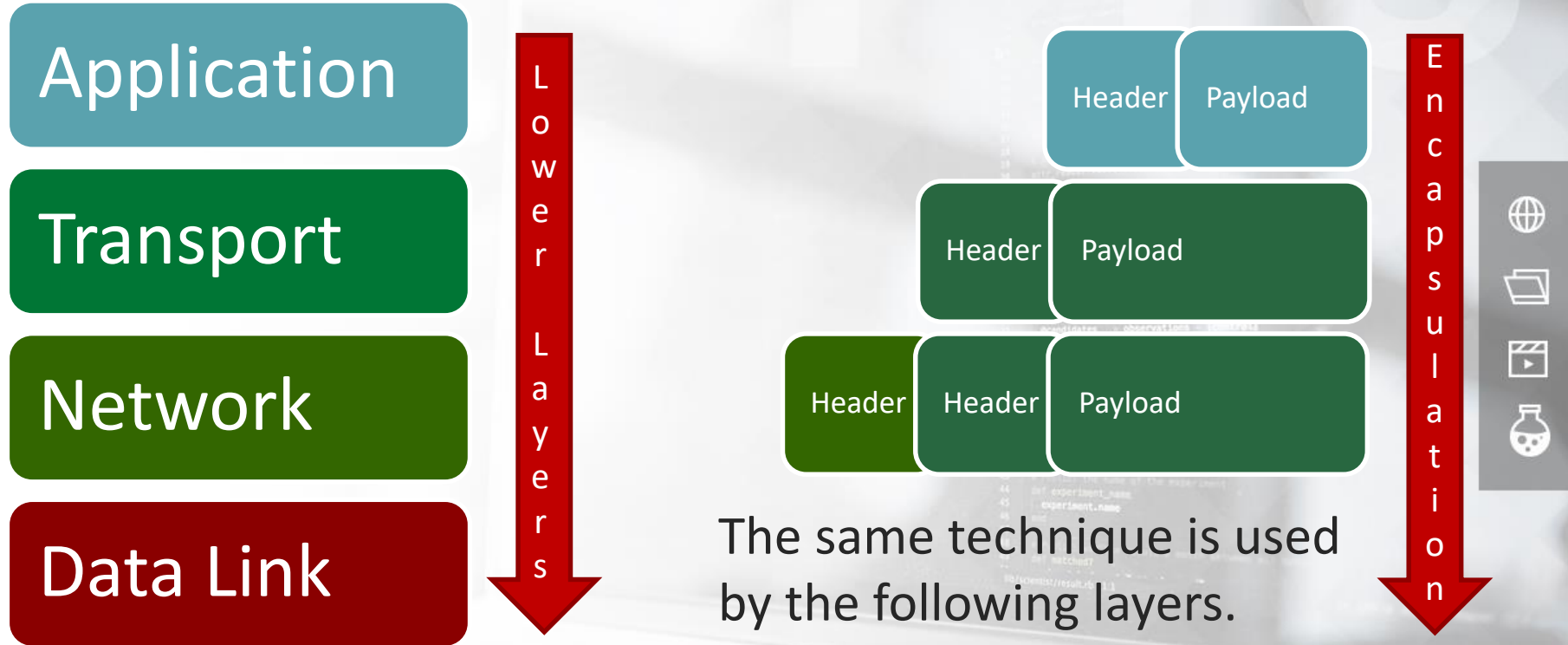
2.1.4 Encapsulation



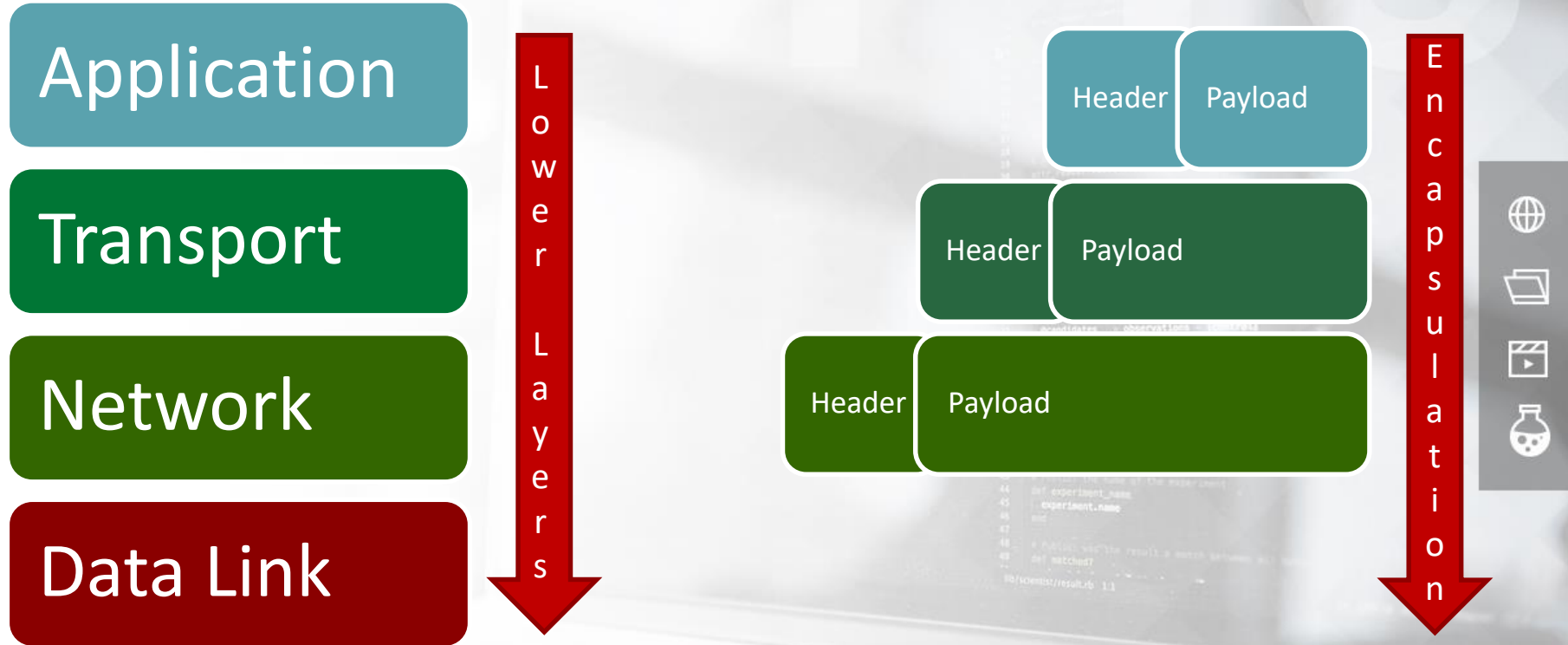
2.1.4 Encapsulation



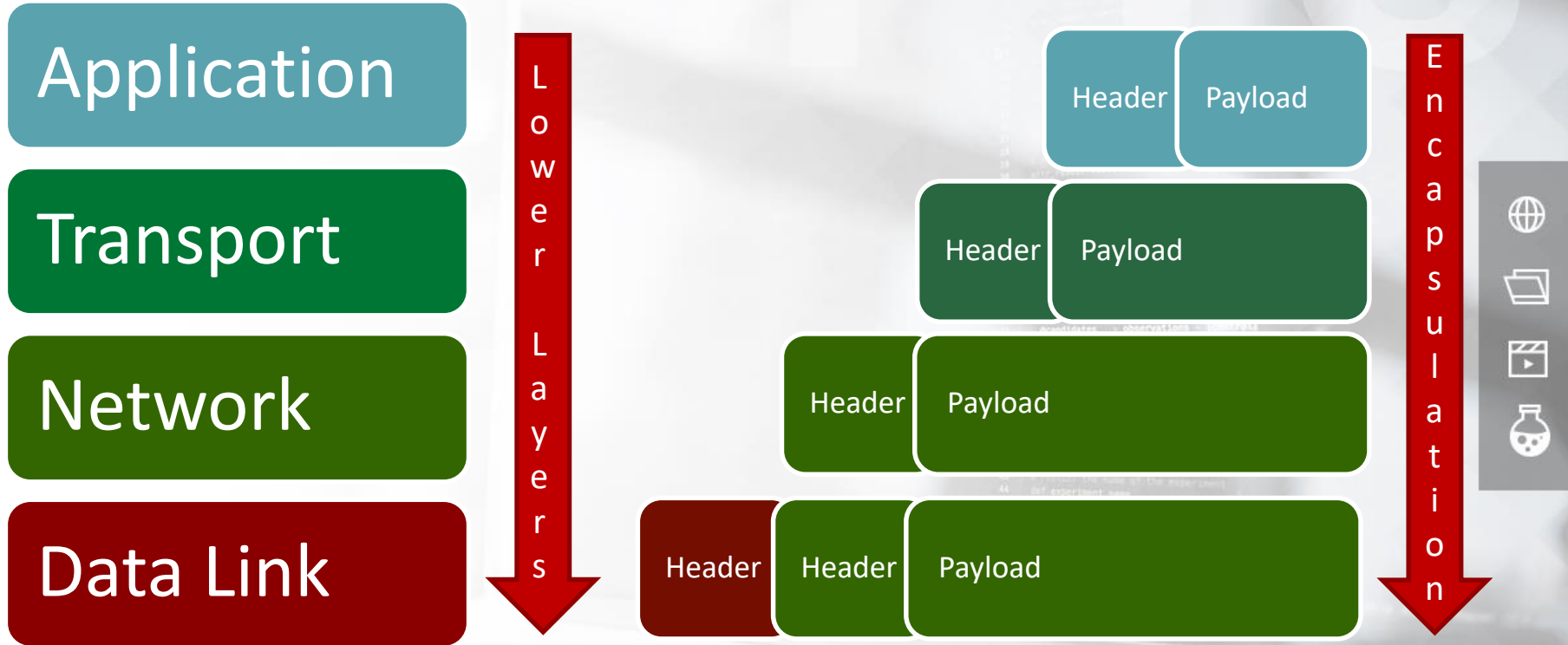
2.1.4 Encapsulation



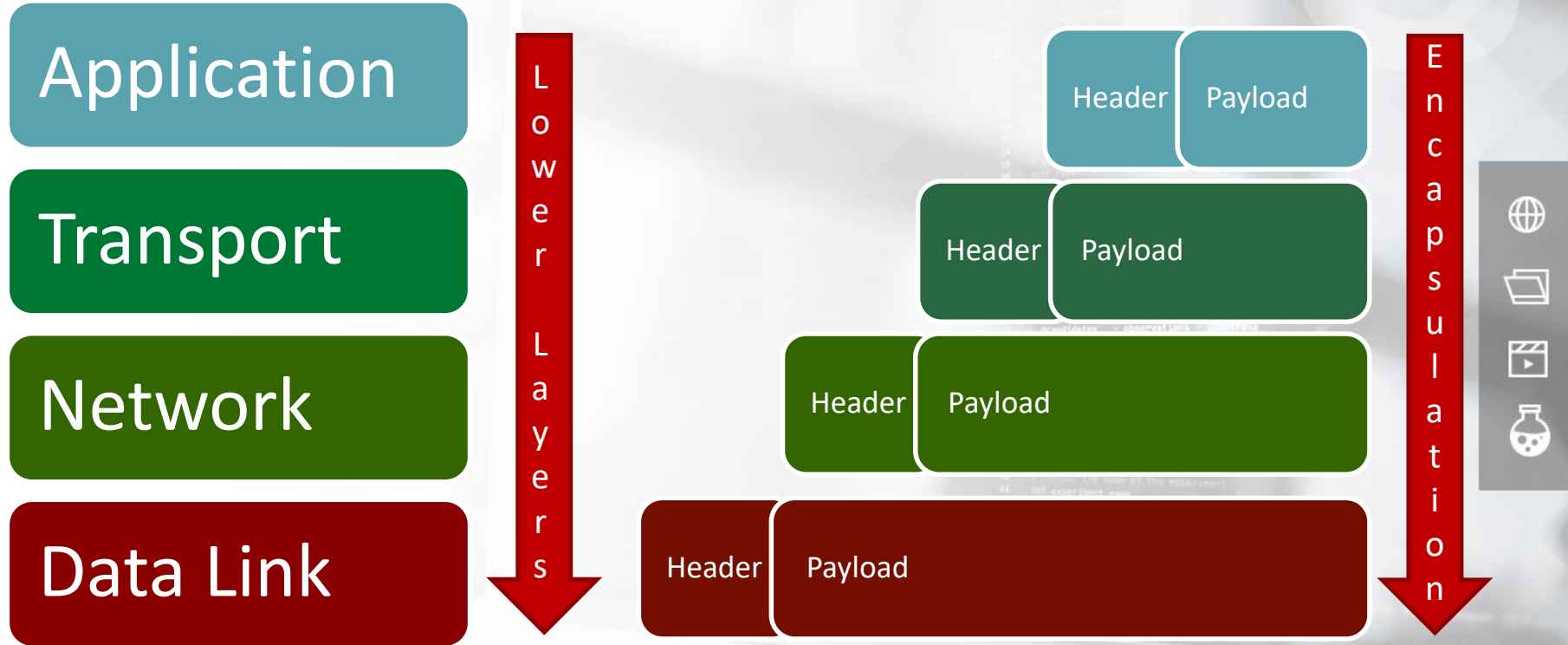
2.1.4 Encapsulation



2.1.4 Encapsulation



2.1.4 Encapsulation



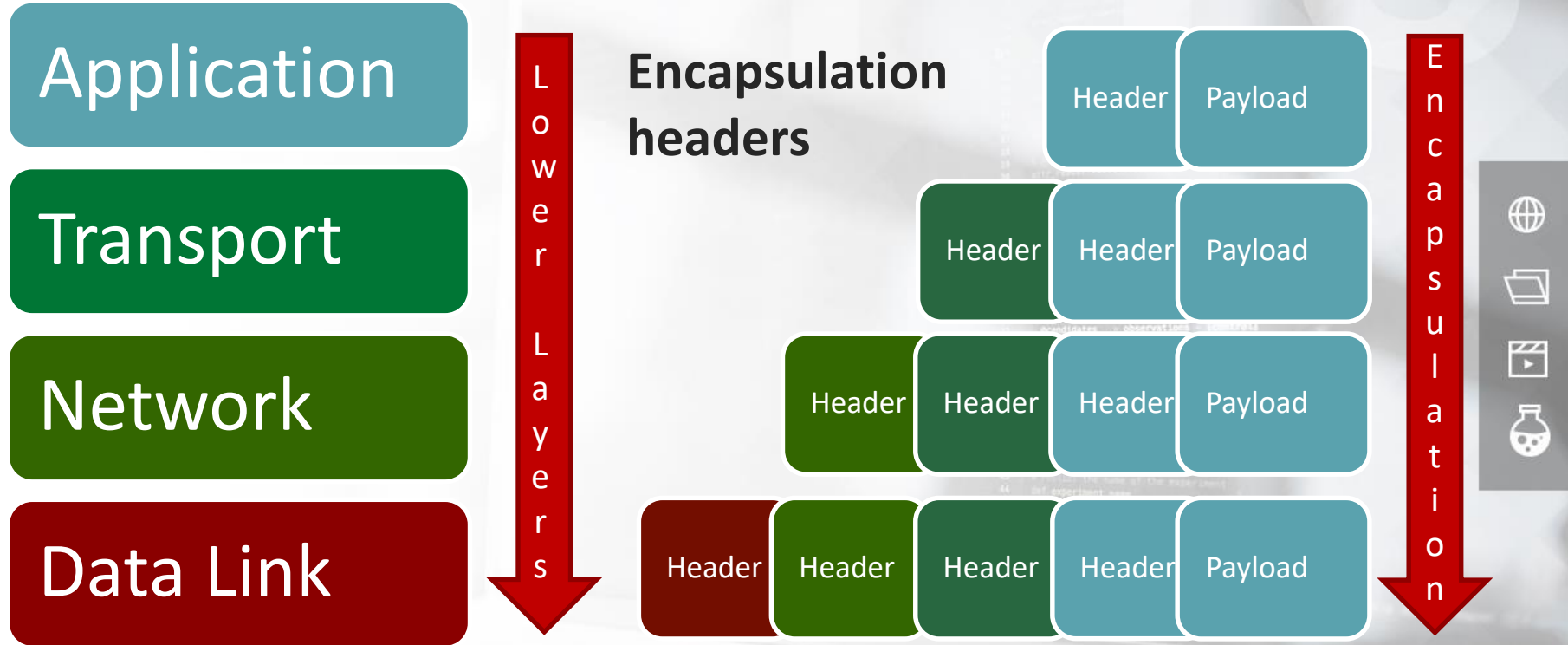
2.1.4 Encapsulation

During encapsulation every protocol adds its own header to the packet, treating it as a payload.

This happens to **every packet** sent by a host.



2.1.4 Encapsulation



2.1.4 Encapsulation

The receiving host does the same operation in reverse order. Using this method, the application does not need to worry about how the transport, network and link layers work. It just hands in the packet to the transport layer.

You will see encapsulation in practice later, during the Wireshark section.



Internet Protocol (IP)



2.2 IP

How does this support my pentesting career?

- Understanding network attacks
- Using network attack tools at their maximum
- Studying other networking protocols

2.2 IP

The **Internet Protocol** (IP) is the protocol that runs on the **Internet** layer of the Internet Protocol suite, also known as TCP/IP.

IP is in charge of delivering the **datagrams** (IP packets are called datagrams) to the hosts involved in a communication, and it uses **IP addresses** to identify a host.



2.2.1 IPv4 Addresses

When you write a letter, you have to specify the recipient's **address** on the envelope before sending it. Similarly, the Internet uses its addressing scheme to deliver packets to the right destination.

Any host on a computer network, be it a private network or the Internet, is identified by a **unique IP address**.



2.2.1 IPv4 Addresses

The vast majority of networks run IP **version 4** (IPv4).

An IPv4 address consists of four bytes, or octets; a byte consists of 8 bits.

73.5.12.132



2.2.1 IPv4 Addresses

A dot delimits every octet in the address.

73.5.12.132

First Second Third Fourth

The diagram shows the IPv4 address 73.5.12.132. Below each of the four numbers (73, 5, 12, 132), there is a red bracket that spans the width of the number. Underneath each bracket is a label: 'First' under 73, 'Second' under 5, 'Third' under 12, and 'Fourth' under 132. This illustrates that each number represents an octet in the address.

2.2.1 IPv4 Addresses

As you may recall from the introduction module, with 8 bits, you can represent **up to 2^8 different values** from 0 to 255.

This does not mean that you can **assign** any address starting from 0.0.0.0 to 255.255.255.255 to a host. Some addresses are **reserved** for special purposes.



2.2.2 Reserved IPv4 Addresses

For example, some reserved intervals are:

- **0.0.0.0 – 0.255.255.255** representing "this" network.
- **127.0.0.0 – 127.255.255.255** representing the local host (e.g., your computer).
- **192.168.0.0 – 192.168.255.255** is reserved for private networks.

You can find the details about the special use of IPv4 addresses in [RFC5735](http://tools.ietf.org/html/rfc5735).

2.2.3 IP/Mask

To fully identify a host, you also need to know its **network**. To do that, you will need an IP address and a **netmask**, or subnet mask.

With an IP/netmask pair, you can identify the network part and the host part of an IP address.

IP address: 192.168.5.100

Subnet mask: 255.255.255.0



2.2.3 IP/Mask

To find the network part you have to perform a **bitwise AND operation** between the netmask and the IP address.

In the following example, we are going to see how to find the network part of this IP address/mask pair:

192.168.33.12/255.255.224.0

2.2.3.1 IP/Mask CIDR Example

1

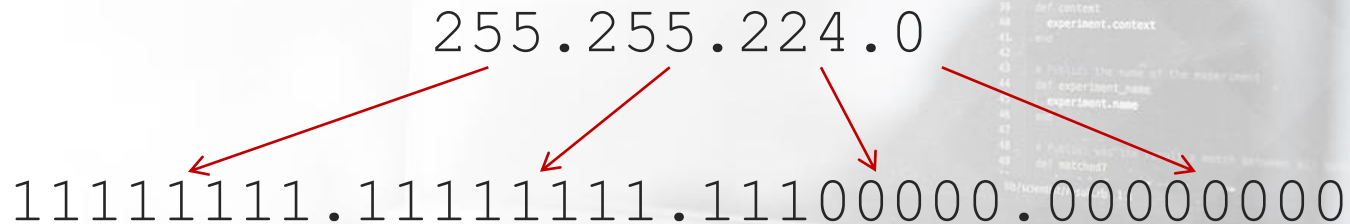
Convert the octets in binary form:

192.168.33.12



11000000.10101000.00100001.00001100

255.255.224.0



11111111.11111111.11100000.00000000

2.2.3.1 IP/Mask CIDR Example

2 Perform the *bitwise AND*:

IP: 11000000.10101000.00100001.00001100

&

Mask: 11111111.11111111.11100000.00000000

=

Network: 11000000.10101000.00100000.00000000

Network prefix in decimal notation:

192.168.32.0

2.2.3.1 IP/Mask CIDR Example

192.168.32.0 is the **network prefix**. You can identify the network by using the following notation:

192.168.32.0/255.255.224.0

Or, as the netmask is made by 19 consecutive "1" bits:

192.168.32.0/19

The latter is the **Classless Inter-Domain Routing (CIDR)** notation.

2.2.3.2 IP/Mask Host Example

The address part not covered by the netmask is the **host part** of the IP address. You can find it by performing a bitwise *AND* with the **inverse of the netmask**.

Let's look at an example with the same IP/mask.



2.2.3.2 IP/Mask Host Example

1

Convert the octets in binary form:

192.168.33.12

11000000.10101000.00100001.00001100

255.255.224.0

11111111.11111111.11100000.00000000

2.2.3.2 IP/Mask Host Example

2

Invert the netmask by performing a *bitwise NOT*:

$$\neg (11111111.11111111.11100000.00000000) \\ = \\ 00000000.00000000.00011111.11111111$$

2.2.3.2 IP/Mask Host Example

3

Perform the final *bitwise AND*:

IP: 11000000.10101000.00100001.00001100

&

¬Mask: 00000000.00000000.00011111.11111111

=

Host: 00000000.00000000.00000001.00001100

Host part in decimal
notation:

0.0.1.12

2.2.3.2 IP/Mask Host Example

Moreover, the inverse of the netmask lets you know how many hosts a network can contain.

In our example, we have 13 bits to represent the hosts; this means that the network can contain $2^{13} = \mathbf{8192}$ different addresses.



2.2.4 Network and Broadcast Addresses

There are two special addresses:

- One with the host part made by all zeros.
- Another with the host part made by all ones.

These special addresses **were** used as the **network** and **broadcast** addresses, thus reducing by 2 the number of hosts on a given network. This technical limitation should be extinct ([RFC1878](http://tools.ietf.org/html/rfc1878)) but is still used to keep compatibility with old equipment.

2.2.5 IP Examples

Let's recap by going over some IP examples.

[illegible]

2.2.5 IP Examples

➤ 10.54.12.0/24 (10.54.12.0/255.255.255.0)

- Contains $2^8 = 256$ addresses
- 10.54.12.0 is the network address according to the pre-CIDR standard
- 10.54.12.255 is the broadcast address according to the pre-CIDR standard



2.2.5 IP Examples

- 192.168.114.32/27 (192.168.114.32/255.255.255.224)
 - Contains $2^5 = 32$ addresses
 - 192.168.114.32 is the pre-CIDR network address
 - 192.168.114.63 is the pre-CIDR broadcast address

2.2.5 IP Examples

Given the network 172.16.2.0/23

- 172.16.3.12 and 172.16.2.66 **are** in the same network
- 172.16.3.240 and 172.16.4.2 **are not** in the same network

The network 192.168.1.0/16

- Does not make sense; a bitwise *AND* between 192.168.1.0 and 255.255.0.0 leads to 192.168.0.0 as network address
- Could be a valid IP address in the 192.168.1.0/16 network

2.2.6 Subnet Calculators

You can practice more on this topic by using a subnet calculator.

Here are two subnet calculators you can check out:

- [A classful calculator](http://www.subnet-calculator.com/)
- [A CIDR calculator](http://www.subnet-calculator.com/cidr.php)

2.2.7 IPv6

IPv4 addresses are being consumed rapidly due to a large number of new devices connecting to the internet every day.

One day IPv4 addresses might be exhausted.



2.2.7 IPv6

As a 32-bit address, **IPv4** has $2^{32} = 4.294.967.296$ possible addresses.

While a 128-bit **IPv6** address has $2^{128} = 2^{32} * 2^{96}$ possible addresses.

2⁹⁶ is equal to **79 octillion** addresses

2.2.7 IPv6

An **IPv6** address consists of **16-bit hexadecimal numbers** separated by a **colon (:)**. Hexadecimal numbers are case insensitive. In case zeros occur, they can be skipped.

Let's check out some IPv6 examples on the next slide.



2.2.7 IPv6

IPv6 addresses examples:

2001:0db8:0020:130F:0000:0000:087C:140B

2001:0db8:0:160F::850C:140B



2.2.7.1 IPv6 header

0								15								16								31															
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7								
Version (4)								Traffic Class (8)								Flow label (20)																							
Payload length (16)																Next header (8)								Hop limit (8)															
Source Address (128)																																							
Destination Address (128)																																							

2.2.7.2 IPv6 forms

IPv6 can be presented in following text representations:

- Regular form: **1080:0:FF:0:8:800:200C:417A**
- Compressed form: **FF01:0:0:0:0:0:0:43** becomes **FF01::43** as a result of skipping zeros
- IPv4-compatible: **0:0:0:0:0:0:13.1.68.3** or **::13.1.68.3** after skipping zeros

2.2.7.3 IPv6 Reserved Addresses

IPv6 also has reserved addresses, which cannot be used like the reserved IPv4 ones.

For example:

- **::1/128 is a loopback address**
- **::FFFF:0:0/96 are IPv4 mapped addresses**

For more information, you can check [RFC3513](https://tools.ietf.org/html/rfc3513)

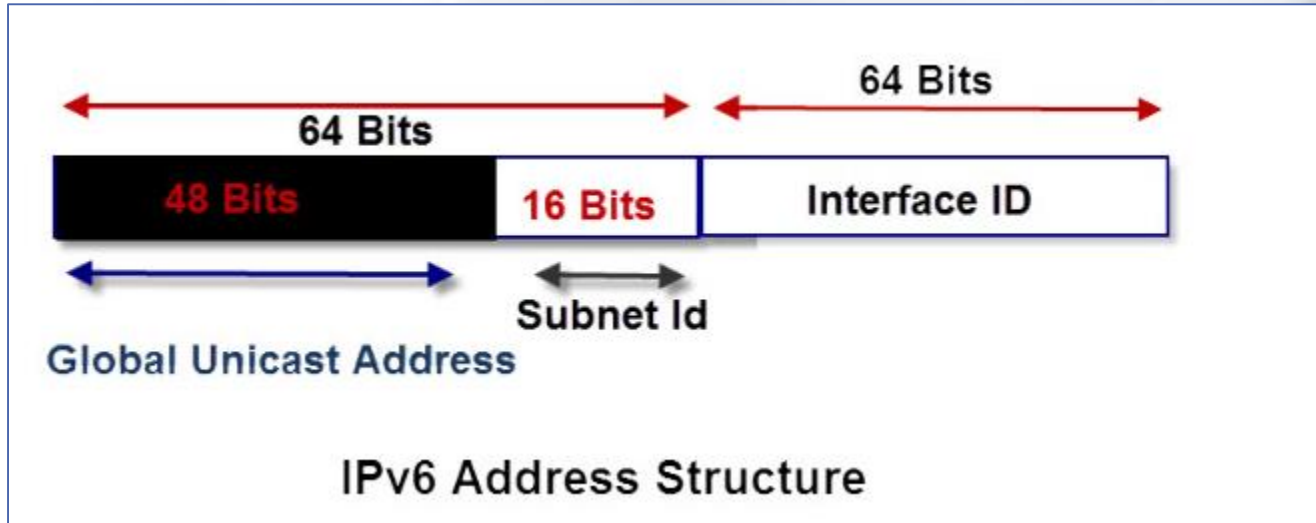
2.2.7.4 IPv6 Structure

An **IPv6 address** can be split in half (64 bits each) into a **network part** and a **device part**.

Furthermore, the **first 64 bits** ends with a **dedicated 16-bits space** (one hex word) that can be used only for **specifying a subnet**.



2.2.7.4 IPv6 Structure



2.2.7.5 IPv6 Scope

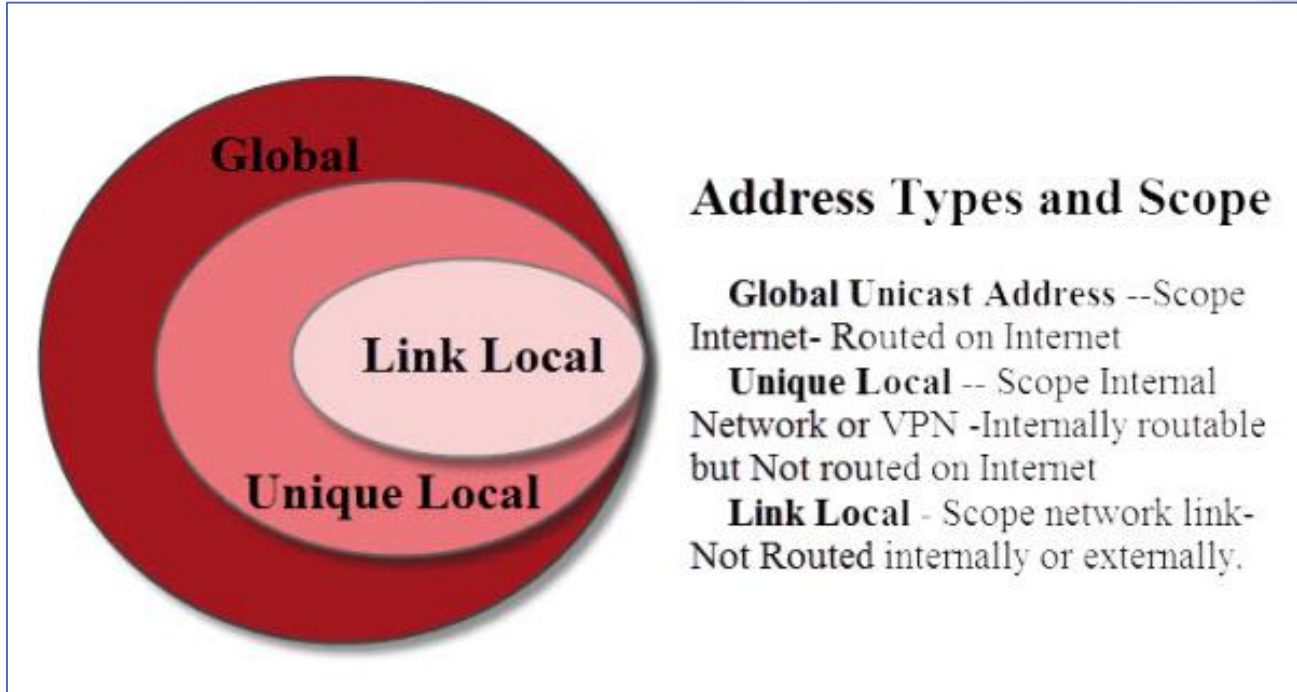
Address Types and Scope

IPv6 addresses have three types:

- **Global Unicast Address** – These addresses are global ones, and reside in global internet.
- **Unique Local and Link Local** – reside only in Internal Networks.



2.2.7.5 IPv6 Scope



2.2.7.6 IPv6 Translation

IPv6 addresses can also be translated to **binary**.

One 4-digit hex word represents **16 binary digits**; we can see this demonstrated in the following way:

- Bin **0000000000000000** = Hex 0000 (or just 0)
- Bin **1111111111111111** = Hex FFFF
- Bin **1101010011011011** = Hex D4DB



2.2.7.6 IPv6 Translation

Thus, 128-bit binary address looks like:

11111111111111111111.11111111111111111111.11111111111111111111
11.11111111111111111111.11111111111111111111.11111111111111111111
11111.11111111111111111111.11111111111111111111

And, the above can be represented by 8 hex words,
separated by colons:

FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF

2.2.7.7 IPv6 Subnets

Like IPv4, an IPv6 address has a network portion and a device portion.

Unlike IPv4, an IPv6 address has a dedicated subnetting portion. On the next few slides, we'll show how the ranges are divided in IPv6.



2.2.7.7 IPv6 Subnets

Network Address Range

In IPv6, the first 48 bits are for Internet global addressing.

```
11111111111111111111.11111111111111111111.11111111111111111111
11.000000000000000000.000000000000000000.000000000000000000
00000.000000000000000000.000000000000000000
```

2.2.7.7 IPv6 Subnets

Subnetting Range

The 16 bits from the 49th to the 64th are for defining subnets.

```
00000000000000000000.00000000000000000000.00000000000000000000
00.111111111111111111.00000000000000000000.00000000000000000000
00000.00000000000000000000.00000000000000000000
```


2.2.7.7 IPv6 Subnets

Device (Interface) Range

The last 64 bits are for device (interface) ID's:

```
00000000000000000000.00000000000000000000.00000000000000000000
00.00000000000000000000.11111111111111111111.11111111111111111111
11111.11111111111111111111.11111111111111111111
```


2.2.7.8 IPv6 Subnetting

In **IPv6**, there are **prefixes** instead of subnets blocks. For example:

2001:1111:1234:1234::/64

In the above IPv6 address, the number after the slash (64) is the **number of bits that is used for a prefix**. Everything behind it can be used for **hosts of the subnet**.

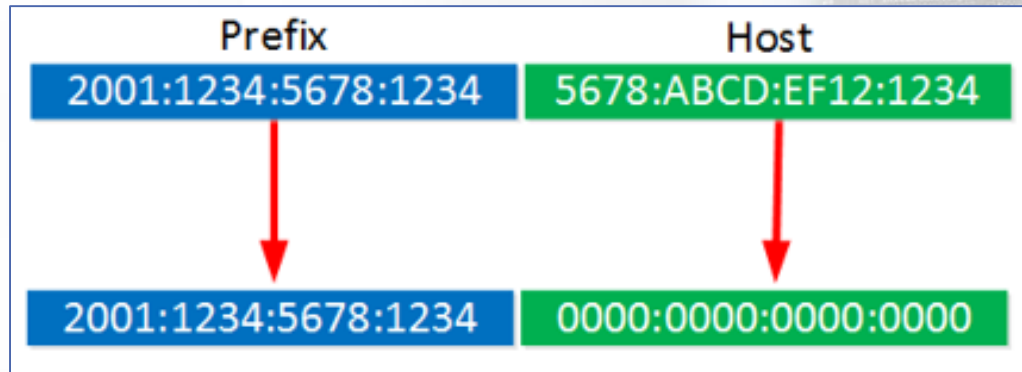
2.2.7.8 IPv6 Subnetting

As you may have noticed, /64 means that the first 64 bits are a prefix. And, as previously mentioned earlier, each 4-digit hex word is 16 bits, thus in following IPv6 address we can divide it as such:

Prefix	Host
2001:1234:5678:1234	5678:ABCD:EF12:1234

2.2.7.8 IPv6 Subnetting

We confirmed that **2001:1234:5678:1234** is the prefix, but let's now focus on writing down a correctly formatted IPv6 address.



2.2.7.8 IPv6 Subnetting

2001:1234:5678:1234:0000:0000:0000:0000 is a valid prefix, but it can be shortened by omitting zeros, into following form:

2001:1234:5678:1234::/64

Routing



2.3 Routing

How does this support my pentesting career?

- Understanding routing protocol attacks
- Performing network traffic inspection

2.3 Routing

Addressing devices is just half of the work needed to reach a host. Your packets need to follow a valid **path** to reach it.

Routers are devices connected to different networks at the same time. They are able to forward IP datagrams from one network to another. The forwarding policy is based on **routing protocols**.



2.3 Routing

Routing protocols are used to determine the best path to reach a network. They behave like a postman who tries to use the shortest path possible to deliver a letter.

A router inspects the destination address of every incoming packet and then forwards it through one of its interfaces.



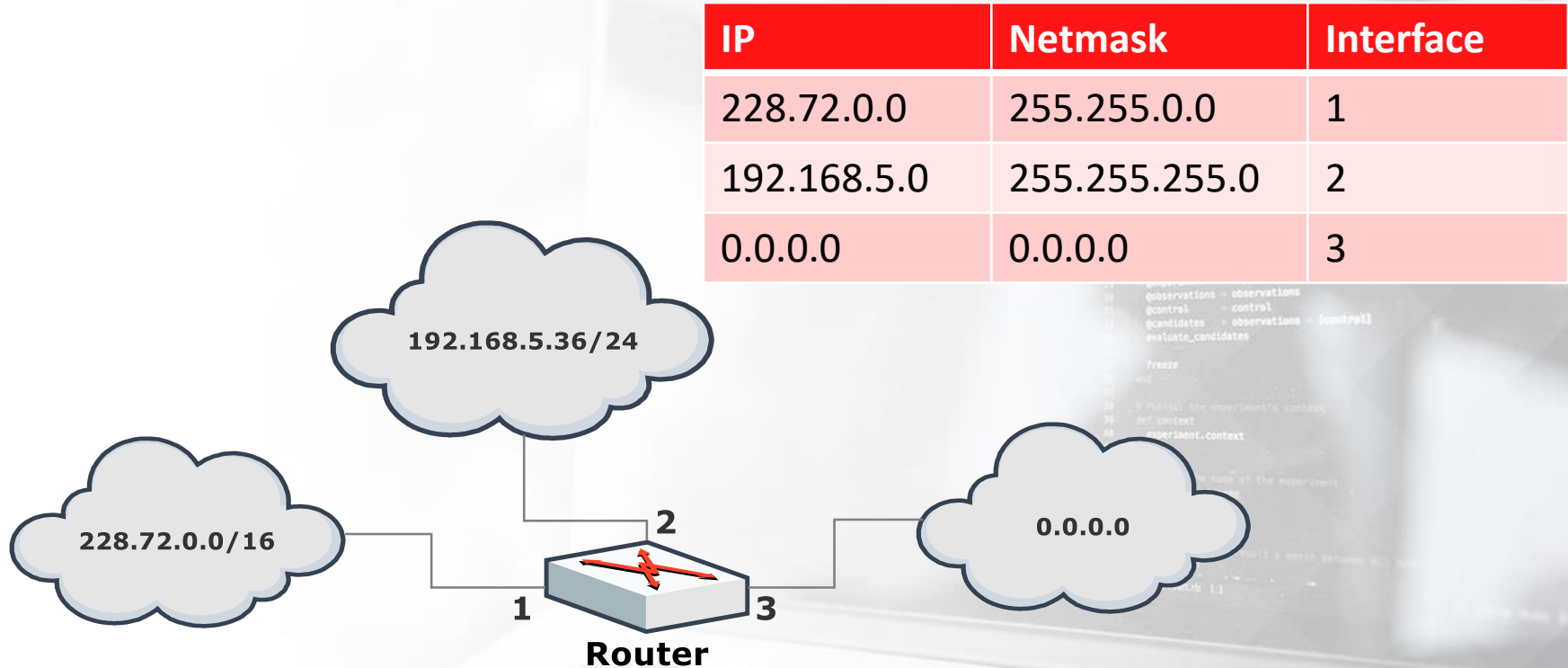
2.3.1 Routing Table

To choose the right forwarding interface, a router performs a lookup in the **routing table**, where it finds an IP-to-interface binding.

The table can also contain an entry with the **default address** (0.0.0.0). This entry is used when the router receives a packet whose destination is an *unknown network*.



2.3.1.1 Routing Table Example



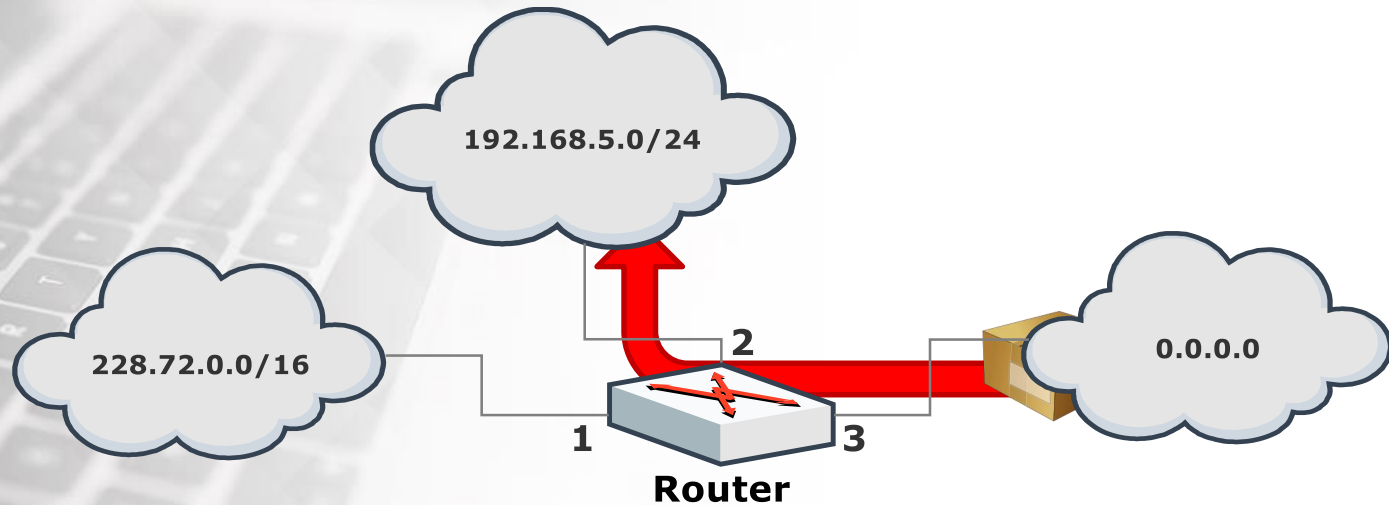
2.3.1.1 Routing Table Example

In this example, the routing table is made of three entries.

- Interface 1 is used to forward the packets to 228.72.0.0/16.
- Interface 2 is used to forward the packets to 192.168.5.0/24.
- Interface 3 is used as the default route for packets whose destination does not match any other entry in the table.

2.3.1.1 Routing Table Example

A packet arriving on interface 3 for 192.168.5.3 is forwarded on interface 2.



2.3.1.1 Routing Table Example

In fact, the first entry in the routing table does not match the destination network.


To: 192.168.5.3



IP	Netmask	Interface
228.72.0.0	255.255.0.0	1
192.168.5.0	255.255.255.0	2
0.0.0.0	0.0.0.0	3



2.3.1.1 Routing Table Example

While the second does: 192.168.5.3 sits in the 192.168.5.0/24 network.



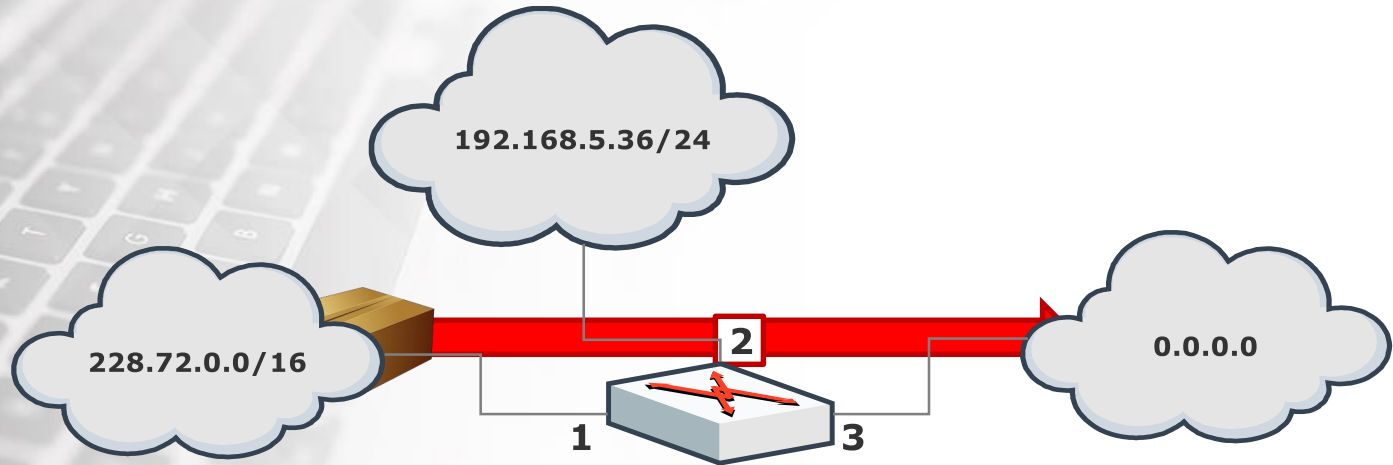
To: 192.168.5.3



IP	Netmask	Interface	
228.72.0.0	255.255.0.0	1	
192.168.5.0	255.255.255.0	2	✓
0.0.0.0	0.0.0.0	3	

2.3.1.2 Default Route Example

A packet arriving on interface 1 for 72.13.37.2 is routed through interface 3, the default route.



Router

2.3.1.2 Default Route Example

There is no matching entry, so the router forwards the packet through interface 3.



To: 72.13.37.2



IP	Netmask	Interface
228.72.0.0	255.255.0.0	1
192.168.5.0	255.255.255.0	2
0.0.0.0	0.0.0.0	3



2.3.2 Routing Metrics

This ensures that, if two paths have the same number of hops, the fastest route is selected.

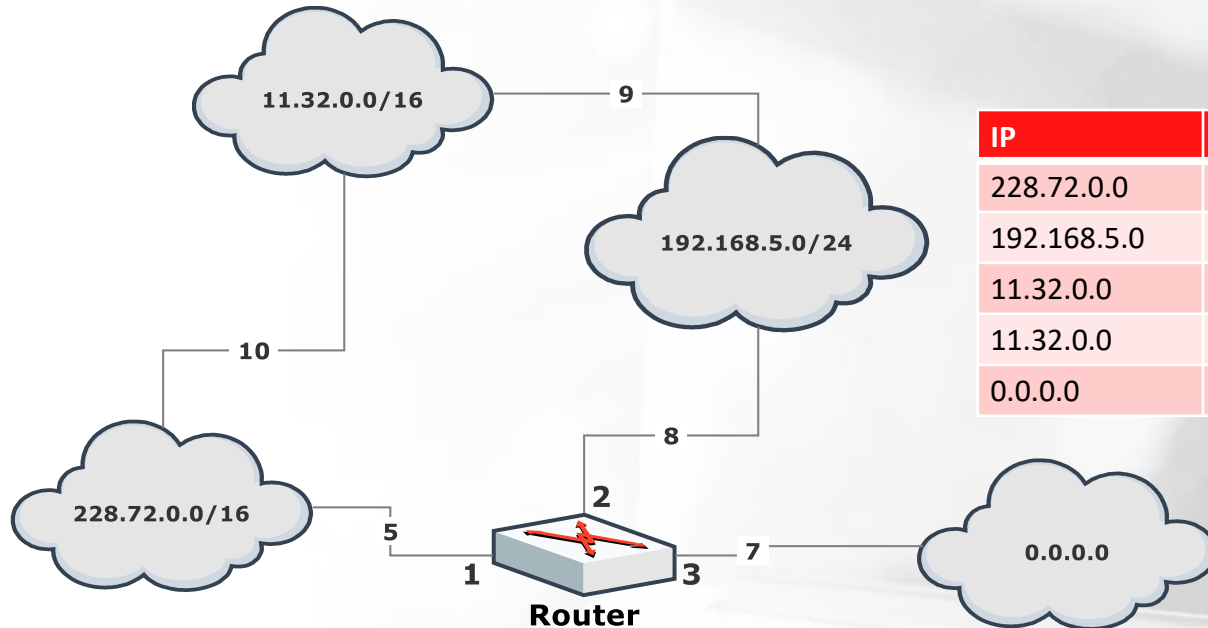
The metric is selected according to the channel's estimated bandwidth and congestion.

```
21 # Optional: Create a new experiment
22
23 # Note: @experiment is the experiment title (which is the
24 # observations - an array of Observations, an array of
25 # @control - the control observation
26
27
28 def initialize(experiment, observations = [], control = nil)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations + [control]
33   evaluate_candidates
34
35   # ...
36
37   # ...
38
39   # ...
40
41   # ...
42
43   # ...
44
45   # ...
46
47   # ...
48
49   # ...
50
51   # ...
52
53   # ...
54
55   # ...
56
57   # ...
58
59   # ...
60
61   # ...
62
63   # ...
64
65   # ...
66
67   # ...
68
69   # ...
70
71   # ...
72
73   # ...
74
75   # ...
76
77   # ...
78
79   # ...
80
81   # ...
82
83   # ...
84
85   # ...
86
87   # ...
88
89   # ...
90
91   # ...
92
93   # ...
94
95   # ...
96
97   # ...
98
99   # ...
100
101   # ...
102
103   # ...
104
105   # ...
106
107   # ...
108
109   # ...
110
111   # ...
112
113   # ...
114
115   # ...
116
117   # ...
118
119   # ...
120
121   # ...
122
123   # ...
124
125   # ...
126
127   # ...
128
129   # ...
130
131   # ...
132
133   # ...
134
135   # ...
136
137   # ...
138
139   # ...
140
141   # ...
142
143   # ...
144
145   # ...
146
147   # ...
148
149   # ...
150
151   # ...
152
153   # ...
154
155   # ...
156
157   # ...
158
159   # ...
160
161   # ...
162
163   # ...
164
165   # ...
166
167   # ...
168
169   # ...
170
171   # ...
172
173   # ...
174
175   # ...
176
177   # ...
178
179   # ...
180
181   # ...
182
183   # ...
184
185   # ...
186
187   # ...
188
189   # ...
190
191   # ...
192
193   # ...
194
195   # ...
196
197   # ...
198
199   # ...
200
201   # ...
202
203   # ...
204
205   # ...
206
207   # ...
208
209   # ...
210
211   # ...
212
213   # ...
214
215   # ...
216
217   # ...
218
219   # ...
220
221   # ...
222
223   # ...
224
225   # ...
226
227   # ...
228
229   # ...
230
231   # ...
232
233   # ...
234
235   # ...
236
237   # ...
238
239   # ...
240
241   # ...
242
243   # ...
244
245   # ...
246
247   # ...
248
249   # ...
250
251   # ...
252
253   # ...
254
255   # ...
256
257   # ...
258
259   # ...
260
261   # ...
262
263   # ...
264
265   # ...
266
267   # ...
268
269   # ...
270
271   # ...
272
273   # ...
274
275   # ...
276
277   # ...
278
279   # ...
280
281   # ...
282
283   # ...
284
285   # ...
286
287   # ...
288
289   # ...
290
291   # ...
292
293   # ...
294
295   # ...
296
297   # ...
298
299   # ...
300
301   # ...
302
303   # ...
304
305   # ...
306
307   # ...
308
309   # ...
310
311   # ...
312
313   # ...
314
315   # ...
316
317   # ...
318
319   # ...
320
321   # ...
322
323   # ...
324
325   # ...
326
327   # ...
328
329   # ...
330
331   # ...
332
333   # ...
334
335   # ...
336
337   # ...
338
339   # ...
340
341   # ...
342
343   # ...
344
345   # ...
346
347   # ...
348
349   # ...
350
351   # ...
352
353   # ...
354
355   # ...
356
357   # ...
358
359   # ...
360
361   # ...
362
363   # ...
364
365   # ...
366
367   # ...
368
369   # ...
370
371   # ...
372
373   # ...
374
375   # ...
376
377   # ...
378
379   # ...
380
381   # ...
382
383   # ...
384
385   # ...
386
387   # ...
388
389   # ...
390
391   # ...
392
393   # ...
394
395   # ...
396
397   # ...
398
399   # ...
400
401   # ...
402
403   # ...
404
405   # ...
406
407   # ...
408
409   # ...
410
411   # ...
412
413   # ...
414
415   # ...
416
417   # ...
418
419   # ...
420
421   # ...
422
423   # ...
424
425   # ...
426
427   # ...
428
429   # ...
430
431   # ...
432
433   # ...
434
435   # ...
436
437   # ...
438
439   # ...
440
441   # ...
442
443   # ...
444
445   # ...
446
447   # ...
448
449   # ...
450
451   # ...
452
453   # ...
454
455   # ...
456
457   # ...
458
459   # ...
460
461   # ...
462
463   # ...
464
465   # ...
466
467   # ...
468
469   # ...
470
471   # ...
472
473   # ...
474
475   # ...
476
477   # ...
478
479   # ...
480
481   # ...
482
483   # ...
484
485   # ...
486
487   # ...
488
489   # ...
490
491   # ...
492
493   # ...
494
495   # ...
496
497   # ...
498
499   # ...
500
501   # ...
502
503   # ...
504
505   # ...
506
507   # ...
508
509   # ...
510
511   # ...
512
513   # ...
514
515   # ...
516
517   # ...
518
519   # ...
520
521   # ...
522
523   # ...
524
525   # ...
526
527   # ...
528
529   # ...
530
531   # ...
532
533   # ...
534
535   # ...
536
537   # ...
538
539   # ...
540
541   # ...
542
543   # ...
544
545   # ...
546
547   # ...
548
549   # ...
550
551   # ...
552
553   # ...
554
555   # ...
556
557   # ...
558
559   # ...
560
561   # ...
562
563   # ...
564
565   # ...
566
567   # ...
568
569   # ...
570
571   # ...
572
573   # ...
574
575   # ...
576
577   # ...
578
579   # ...
580
581   # ...
582
583   # ...
584
585   # ...
586
587   # ...
588
589   # ...
590
591   # ...
592
593   # ...
594
595   # ...
596
597   # ...
598
599   # ...
600
601   # ...
602
603   # ...
604
605   # ...
606
607   # ...
608
609   # ...
610
611   # ...
612
613   # ...
614
615   # ...
616
617   # ...
618
619   # ...
620
621   # ...
622
623   # ...
624
625   # ...
626
627   # ...
628
629   # ...
630
631   # ...
632
633   # ...
634
635   # ...
636
637   # ...
638
639   # ...
640
641   # ...
642
643   # ...
644
645   # ...
646
647   # ...
648
649   # ...
650
651   # ...
652
653   # ...
654
655   # ...
656
657   # ...
658
659   # ...
660
661   # ...
662
663   # ...
664
665   # ...
666
667   # ...
668
669   # ...
670
671   # ...
672
673   # ...
674
675   # ...
676
677   # ...
678
679   # ...
680
681   # ...
682
683   # ...
684
685   # ...
686
687   # ...
688
689   # ...
690
691   # ...
692
693   # ...
694
695   # ...
696
697   # ...
698
699   # ...
700
701   # ...
702
703   # ...
704
705   # ...
706
707   # ...
708
709   # ...
710
711   # ...
712
713   # ...
714
715   # ...
716
717   # ...
718
719   # ...
720
721   # ...
722
723   # ...
724
725   # ...
726
727   # ...
728
729   # ...
730
731   # ...
732
733   # ...
734
735   # ...
736
737   # ...
738
739   # ...
740
741   # ...
742
743   # ...
744
745   # ...
746
747   # ...
748
749   # ...
750
751   # ...
752
753   # ...
754
755   # ...
756
757   # ...
758
759   # ...
760
761   # ...
762
763   # ...
764
765   # ...
766
767   # ...
768
769   # ...
770
771   # ...
772
773   # ...
774
775   # ...
776
777   # ...
778
779   # ...
780
781   # ...
782
783   # ...
784
785   # ...
786
787   # ...
788
789   # ...
790
791   # ...
792
793   # ...
794
795   # ...
796
797   # ...
798
799   # ...
800
801   # ...
802
803   # ...
804
805   # ...
806
807   # ...
808
809   # ...
810
811   # ...
812
813   # ...
814
815   # ...
816
817   # ...
818
819   # ...
820
821   # ...
822
823   # ...
824
825   # ...
826
827   # ...
828
829   # ...
830
831   # ...
832
833   # ...
834
835   # ...
836
837   # ...
838
839   # ...
840
841   # ...
842
843   # ...
844
845   # ...
846
847   # ...
848
849   # ...
850
851   # ...
852
853   # ...
854
855   # ...
856
857   # ...
858
859   # ...
860
861   # ...
862
863   # ...
864
865   # ...
866
867   # ...
868
869   # ...
870
871   # ...
872
873   # ...
874
875   # ...
876
877   # ...
878
879   # ...
880
881   # ...
882
883   # ...
884
885   # ...
886
887   # ...
888
889   # ...
890
891   # ...
892
893   # ...
894
895   # ...
896
897   # ...
898
899   # ...
900
901   # ...
902
903   # ...
904
905   # ...
906
907   # ...
908
909   # ...
910
911   # ...
912
913   # ...
914
915   # ...
916
917   # ...
918
919   # ...
920
921   # ...
922
923   # ...
924
925   # ...
926
927   # ...
928
929   # ...
930
931   # ...
932
933   # ...
934
935   # ...
936
937   # ...
938
939   # ...
940
941   # ...
942
943   # ...
944
945   # ...
946
947   # ...
948
949   # ...
950
951   # ...
952
953   # ...
954
955   # ...
956
957   # ...
958
959   # ...
960
961   # ...
962
963   # ...
964
965   # ...
966
967   # ...
968
969   # ...
970
971   # ...
972
973   # ...
974
975   # ...
976
977   # ...
978
979   # ...
980
981   # ...
982
983   # ...
984
985   # ...
986
987   # ...
988
989   # ...
990
991   # ...
992
993   # ...
994
995   # ...
996
997   # ...
998
999   # ...
1000
```



2.3.2.1 Routing Metrics Example

Let's look at how routing decisions are made according to metrics.

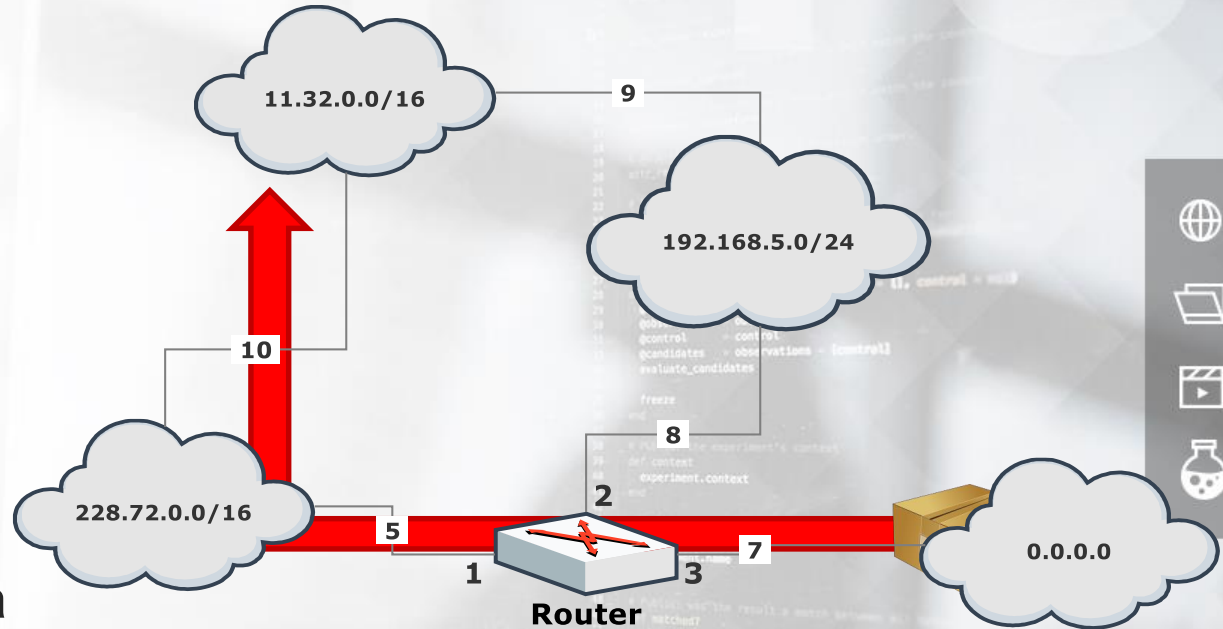


IP	Netmask	Interface	Metric
228.72.0.0	255.255.0.0	1	5
192.168.5.0	255.255.255.0	2	8
11.32.0.0	255.255.0.0	2	17
11.32.0.0	255.255.0.0	1	15
0.0.0.0	0.0.0.0	3	7

2.3.2.1 Routing Metrics Example

A packet arriving on interface 3 for 11.32.3.118 is routed through interface 1, as the metric for that route is 15.

Routing through interface 2 would have a metric of 17.



2.3.3 Checking the Routing Table

Routing tables are not only kept by routers; every host stores its own table.

To check what they look like, you can use:

- `ip route` on Linux
- `route print` on Windows
- `netstat -r` on OSX

```
24 # def initialize_experiment
25 # def initialize_experiment
26 # def initialize_experiment
27 # def initialize_experiment
28 # def initialize_experiment
29 # def initialize_experiment
30 # def initialize_experiment
31 # def initialize_experiment
32 # def initialize_experiment
33 # def initialize_experiment
34 # def initialize_experiment
35 # def initialize_experiment
36 # def initialize_experiment
37 # def initialize_experiment
38 # def initialize_experiment
39 # def initialize_experiment
40 # def initialize_experiment
41 # def initialize_experiment
42 # def initialize_experiment
43 # def initialize_experiment
44 # def initialize_experiment
45 # def initialize_experiment
46 # def initialize_experiment
47 # def initialize_experiment
48 # def initialize_experiment
49 # def initialize_experiment
50 # def initialize_experiment
51 # def initialize_experiment
52 # def initialize_experiment
53 # def initialize_experiment
54 # def initialize_experiment
55 # def initialize_experiment
56 # def initialize_experiment
57 # def initialize_experiment
58 # def initialize_experiment
59 # def initialize_experiment
60 # def initialize_experiment
61 # def initialize_experiment
62 # def initialize_experiment
63 # def initialize_experiment
64 # def initialize_experiment
65 # def initialize_experiment
66 # def initialize_experiment
67 # def initialize_experiment
68 # def initialize_experiment
69 # def initialize_experiment
70 # def initialize_experiment
71 # def initialize_experiment
72 # def initialize_experiment
73 # def initialize_experiment
74 # def initialize_experiment
75 # def initialize_experiment
76 # def initialize_experiment
77 # def initialize_experiment
78 # def initialize_experiment
79 # def initialize_experiment
80 # def initialize_experiment
81 # def initialize_experiment
82 # def initialize_experiment
83 # def initialize_experiment
84 # def initialize_experiment
85 # def initialize_experiment
86 # def initialize_experiment
87 # def initialize_experiment
88 # def initialize_experiment
89 # def initialize_experiment
90 # def initialize_experiment
91 # def initialize_experiment
92 # def initialize_experiment
93 # def initialize_experiment
94 # def initialize_experiment
95 # def initialize_experiment
96 # def initialize_experiment
97 # def initialize_experiment
98 # def initialize_experiment
99 # def initialize_experiment
100 # def initialize_experiment
```



2.3.3 Checking the Routing Table

Checking the routing table on a Linux box:



```
root@host:~# ip route
default via 192.168.51.1 dev eth0 proto static
192.168.51.0/24 dev wlan0 proto kernel scope link src 192.168.51.123
```



2.3.3 Checking the Routing Table

Checking the routing table on Microsoft Windows:



```
C:\Users\User>route print
```

```
=====
```

```
Interface List
```

```
11...08 00 27 bf ac c8 .....Intel(R) PRO/1000 MT Desktop Adapter
```

```
1.....Software Loopback Interface 1
```

```
=====
```

```
IPv4 Route Table
```

```
=====
```

```
Active Routes:
```

Network	Destination	Netmask	Gateway	Interface	Metric
	0.0.0.0	0.0.0.0	10.0.2.2	10.0.2.15	10
	10.0.2.0	255.255.255.0	On-link	10.0.2.15	266

2.3.3 Checking the Routing Table

Checking the routing table on Mac OSX:



```
User:~ user$ netstat -r
```

```
Routing tables
```

```
Internet:
```

Destination	Gateway	Flags	Refs	Use	Netif	Expire
default	192.168.51.1	UGSc	13	0	en1	
127	127.0.0.1	UCS	0	0	lo0	
127.0.0.1	127.0.0.1	UH	1	16	lo0	
169.254	link#4	UCS	0	0	en1	
192.168.51	link#4	UCS	4	0	en1	
192.168.51.1	58:6d:8f:e5:e:d2	UHLWIir	14	24	en1	1200
192.168.51.109	2:f:b5:4b:76:cf	UHLWii	0	0	en1	1148



Link Layer Devices and Protocols



2.4 Link Layer Devices and Protocols

How does this support my pentesting career?

- MAC spoofing
- Testing switches security
- Sniffing techniques
- Man in the middle attacks

2.4 Link Layer Devices and Protocols

Packet forwarding also happens in the **lowest layer** of the TCP/IP stack: the **link layer**.

While routers are aware of the best overall path to the destination, link layer devices and protocols deal only with the next hop.



2.4 Link Layer Devices and Protocols

In this section you will see:

- How switches work
- Network card's MAC addresses
- The Address Resolution Protocol (ARP)

```
1 # Create a new experiment
2 def create_experiment():
3     # Create a new experiment
4     # Create a new experiment
5     # Create a new experiment
6     # Create a new experiment
7     # Create a new experiment
8     # Create a new experiment
9     # Create a new experiment
10    # Create a new experiment
11    # Create a new experiment
12    # Create a new experiment
13    # Create a new experiment
14    # Create a new experiment
15    # Create a new experiment
16    # Create a new experiment
17    # Create a new experiment
18    # Create a new experiment
19    # Create a new experiment
20    # Create a new experiment
21    # Create a new experiment
22    # Create a new experiment
23    # Create a new experiment
24    # Create a new experiment
25    # Create a new experiment
26    # Create a new experiment
27    # Create a new experiment
28    # Create a new experiment
29    # Create a new experiment
30    # Create a new experiment
31    # Create a new experiment
32    # Create a new experiment
33    # Create a new experiment
34    # Create a new experiment
35    # Create a new experiment
36    # Create a new experiment
37    # Create a new experiment
38    # Create a new experiment
39    # Create a new experiment
40    # Create a new experiment
41    # Create a new experiment
42    # Create a new experiment
43    # Create a new experiment
44    # Create a new experiment
45    # Create a new experiment
46    # Create a new experiment
47    # Create a new experiment
48    # Create a new experiment
49    # Create a new experiment
50    # Create a new experiment
51    # Create a new experiment
52    # Create a new experiment
53    # Create a new experiment
54    # Create a new experiment
55    # Create a new experiment
56    # Create a new experiment
57    # Create a new experiment
58    # Create a new experiment
59    # Create a new experiment
60    # Create a new experiment
61    # Create a new experiment
62    # Create a new experiment
63    # Create a new experiment
64    # Create a new experiment
65    # Create a new experiment
66    # Create a new experiment
67    # Create a new experiment
68    # Create a new experiment
69    # Create a new experiment
70    # Create a new experiment
71    # Create a new experiment
72    # Create a new experiment
73    # Create a new experiment
74    # Create a new experiment
75    # Create a new experiment
76    # Create a new experiment
77    # Create a new experiment
78    # Create a new experiment
79    # Create a new experiment
80    # Create a new experiment
81    # Create a new experiment
82    # Create a new experiment
83    # Create a new experiment
84    # Create a new experiment
85    # Create a new experiment
86    # Create a new experiment
87    # Create a new experiment
88    # Create a new experiment
89    # Create a new experiment
90    # Create a new experiment
91    # Create a new experiment
92    # Create a new experiment
93    # Create a new experiment
94    # Create a new experiment
95    # Create a new experiment
96    # Create a new experiment
97    # Create a new experiment
98    # Create a new experiment
99    # Create a new experiment
100   # Create a new experiment
```

2.4.1 Link Layer Devices

Hubs and switches are network devices that forward **frames** (layer 2 packets) on a local network.

They work with link layer network addresses: **MAC addresses**.



2.4.2 Mac Addresses

IP addresses are the Layer 3 (Network layer) addressing scheme used to identify a host in a network, while **MAC addresses** uniquely identify a network card (Layer 2).

A MAC (Media Access Control) address is also known as the **physical address**.



2.4.2 Mac Addresses

MAC addresses are 48 bit (6 bytes) long and are expressed in hexadecimal form (HEX).

00:11:AA:22:EE:FF



2.4.2 Mac Addresses

Every host on a network has both a MAC and an IP address.

Let us see how they are used together to send packets.

Remember: the lower layer serves the layer above.

...

IP Layer

Link Layer



2.4.3 IP and MAC Addresses

Let's take a look at an example to see how MAC addresses are used.



2.4.3 IP and MAC Addresses

Two different networks are connected together by a router:

- 10.32.1.0/24
- 192.168.2.0/24



2.4.3 IP and MAC Addresses

Every host on the network has both an IP and a MAC address. The router has two interfaces, each with its own addresses.



2.4.3 IP and MAC Addresses

If workstation A wants to send a packet to workstation B, which IP and MAC addresses will it use?



2.4.3 IP and MAC Addresses

Workstation A will create a packet with:

- The **destination IP address of workstation B** in the IP header of the datagram.
- The **destination MAC address of the router** in the link layer header of the frame.
- The **source IP address of workstation A**
- The **source MAC address of workstation A**

2.4.3 IP and MAC Addresses

The router will then take the packet and forward it to B's network, **rewriting the packet's MAC addresses**:

- The **destination MAC address** will be B's
- The **source MAC address** will be the router's

The router will not change the source and destination IP addresses.



2.4.3 IP and MAC Addresses

When a device sends a packet:

- The destination MAC address is the MAC address of the **next hop**; this ensures that, locally, the network knows where to forward the packet.
- The destination IP address is the address of the **destination host**; this is global information and remains the same along the packet trip.

2.4.3 IP and MAC Addresses

This method, in a way, recalls how you send a letter to a friend.

You need to know his or her home address (IP address) and the address of the nearest post office (MAC address) where you can drop the letter.

2.4.4 Broadcast MAC Address

There is also a special MAC address

FF:FF:FF:FF:FF:FF

...which is the **broadcast** MAC address.

A frame (the name of the packets at Layer 2) with this address is delivered to all the hosts in the local network (within the same broadcast domain).



2.4.5 Switches

While routers work with IP addresses, switches work with MAC addresses. Switches also have multiple interfaces, so they need to keep a **forwarding table** that binds one or more MAC addresses to an interface.



MAC	Interface	TTL
00:11:22:33:44:55	1	30
AA:BB:CC:DD:EE:01	2	5
AA:CC:FF:0A:0C:12	2	5
11:22:33:1D:CC:0A	3	7

2.4.5 Switches

The forwarding table is called Content Addressable Memory (CAM) table. Many hosts can connect to a switch. Let's see how.



MAC	Interface	TTL
00:11:22:33:44:55	1	30
AA:BB:CC:DD:EE:01	2	5
AA:CC:FF:0A:0C:12	2	5
11:22:33:1D:CC:0A	3	7

2.4.5 Switches

The smallest switches you can encounter are home switches, usually integrated into a DSL home router. They usually have 4 ports.

Corporate switches may have up to 64 ports, and system administrators can connect multiple switches together to accommodate more hosts.

2.4.5 Switches

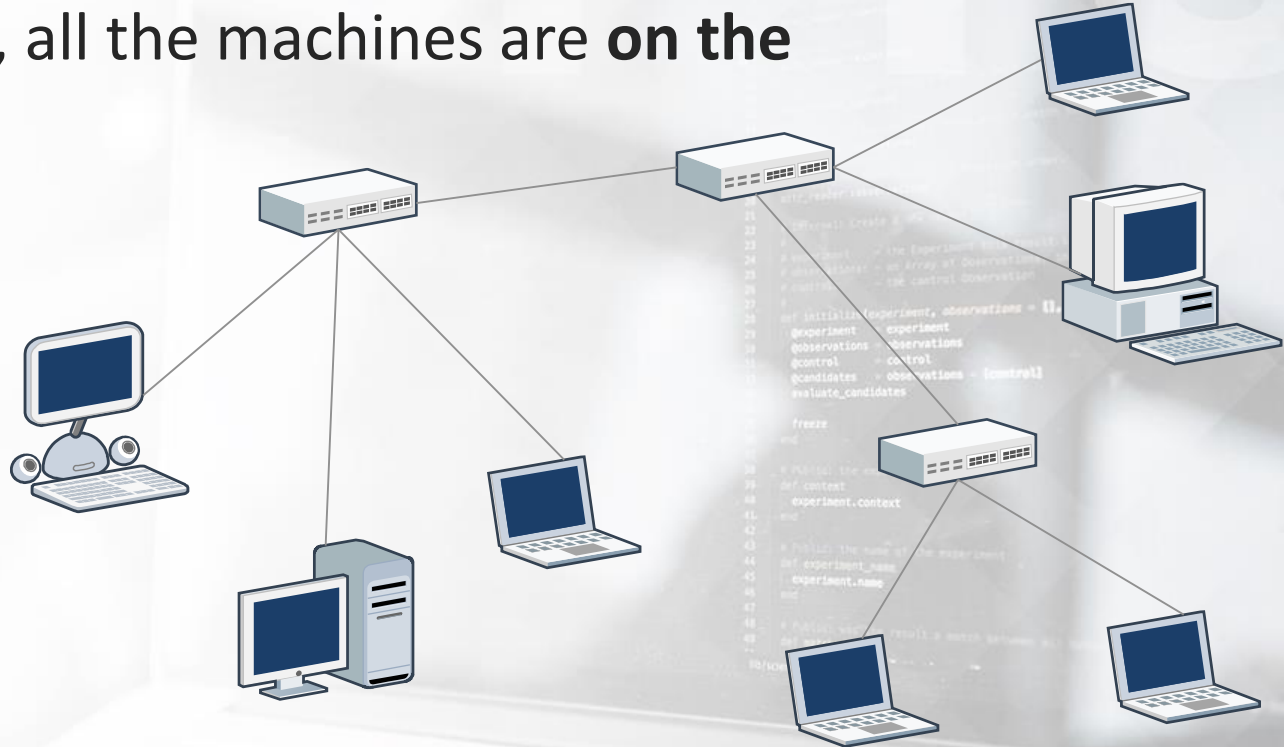
The main difference between one switch and another is the packet forwarding speed.

The speed of a switch varies from 10Mbps (megabits per second) to 10Gbps (gigabits per second). Nowadays, 1Gbps is the most common forwarding speed in commercial switches.



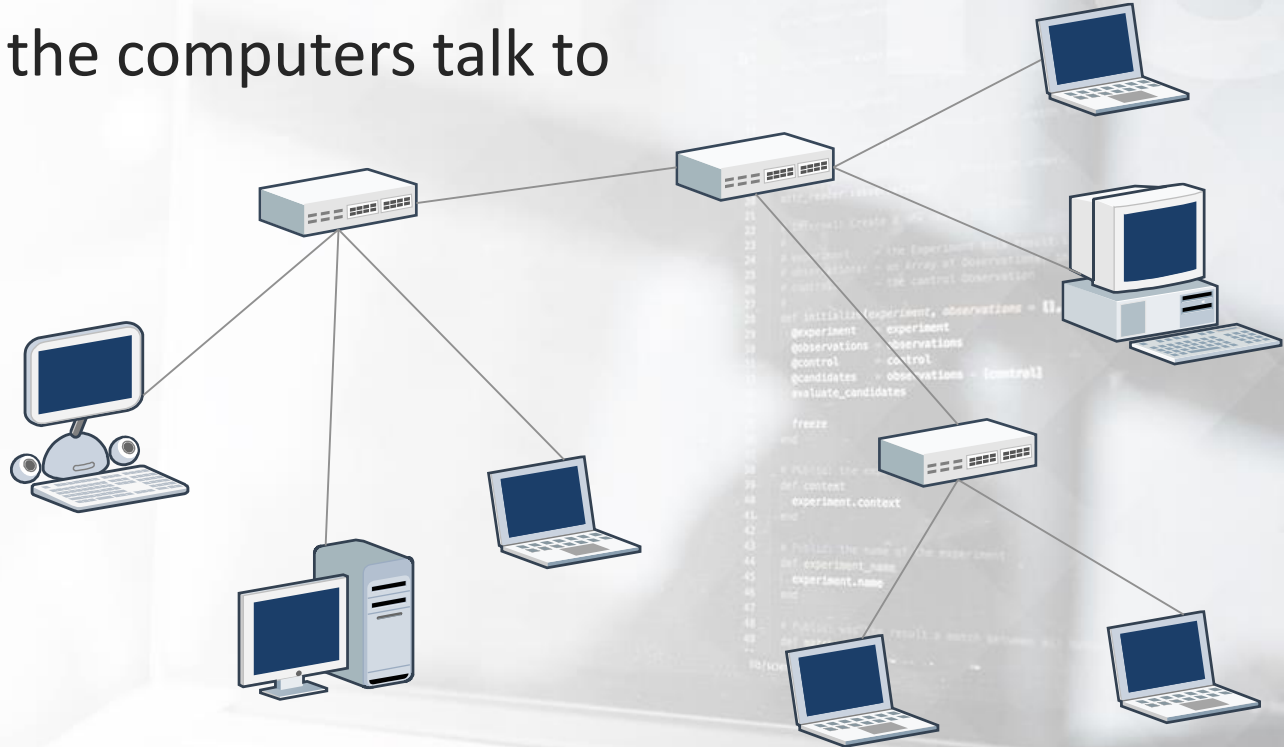
2.4.5.1 Multi-switch Network

In this diagram, all the machines are **on the same network**.



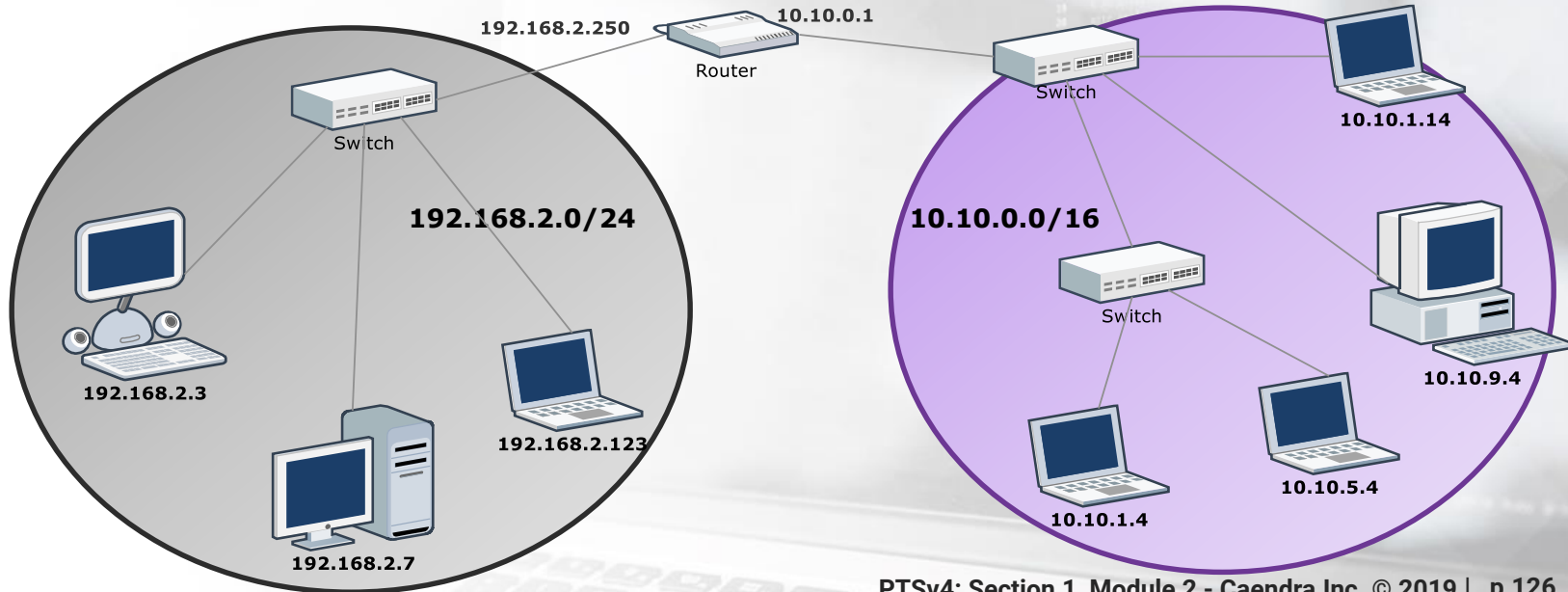
2.4.5.1 Multi-switch Network

Switches let all the computers talk to each other.



2.4.5.2 Segmentation

Switches, without VLANs, do not **segment** networks. Routers do.



2.4.5.2 Segmentation

Usually, every interface of a router is attached to a different subnet with a different network address.

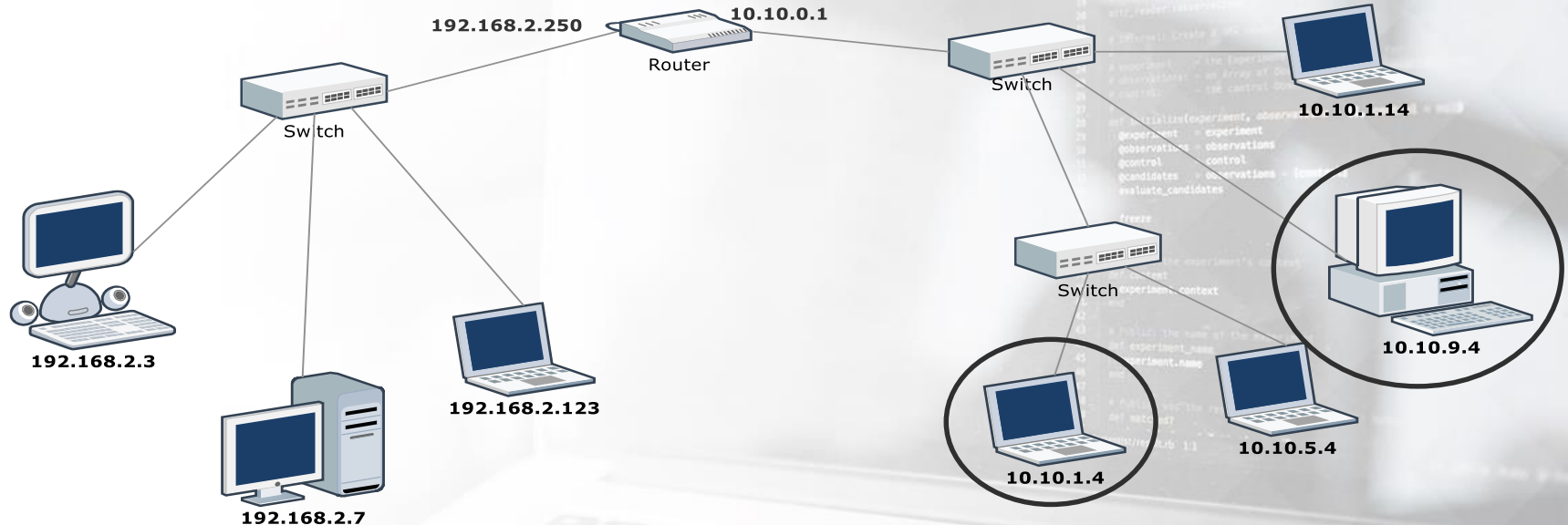
Also, routers do not forward packets coming from one interface if they have a ff:ff:ff:ff:ff:ff broadcast MAC address (imagine if they did!).

```
18 def _create_observation(self, experiment, observations):
19     """Create a new observation for the experiment"""
20     # Create a new observation
21     observation = Observation(experiment.name, self._create_observation_data(experiment, observations))
22     # Add the observation to the list of observations
23     observations.append(observation)
24     # Return the list of observations
25     return observations
26
27 def _create_observation_data(self, experiment, observations):
28     """Create a new observation data for the experiment"""
29     # Create a new observation data
30     observation_data = {}
31     # Add the experiment name to the observation data
32     observation_data['experiment_name'] = experiment.name
33     # Add the observation data to the list of observation data
34     observation_data['observations'] = observations
35     # Return the observation data
36     return observation_data
37
38 def _create_observation_context(self, experiment, observations):
39     """Create a new observation context for the experiment"""
40     # Create a new observation context
41     observation_context = {}
42     # Add the experiment name to the observation context
43     observation_context['experiment_name'] = experiment.name
44     # Add the observation context to the list of observation context
45     observation_context['observations'] = observations
46     # Return the observation context
47     return observation_context
48
49 def _create_observation_result(self, experiment, observations):
50     """Create a new observation result for the experiment"""
51     # Create a new observation result
52     observation_result = {}
53     # Add the experiment name to the observation result
54     observation_result['experiment_name'] = experiment.name
55     # Add the observation result to the list of observation result
56     observation_result['observations'] = observations
57     # Return the observation result
58     return observation_result
59
60 def _create_observation_result_sub(self, experiment, observations):
61     """Create a new observation result sub for the experiment"""
62     # Create a new observation result sub
63     observation_result_sub = {}
64     # Add the experiment name to the observation result sub
65     observation_result_sub['experiment_name'] = experiment.name
66     # Add the observation result sub to the list of observation result sub
67     observation_result_sub['observations'] = observations
68     # Return the observation result sub
69     return observation_result_sub
```



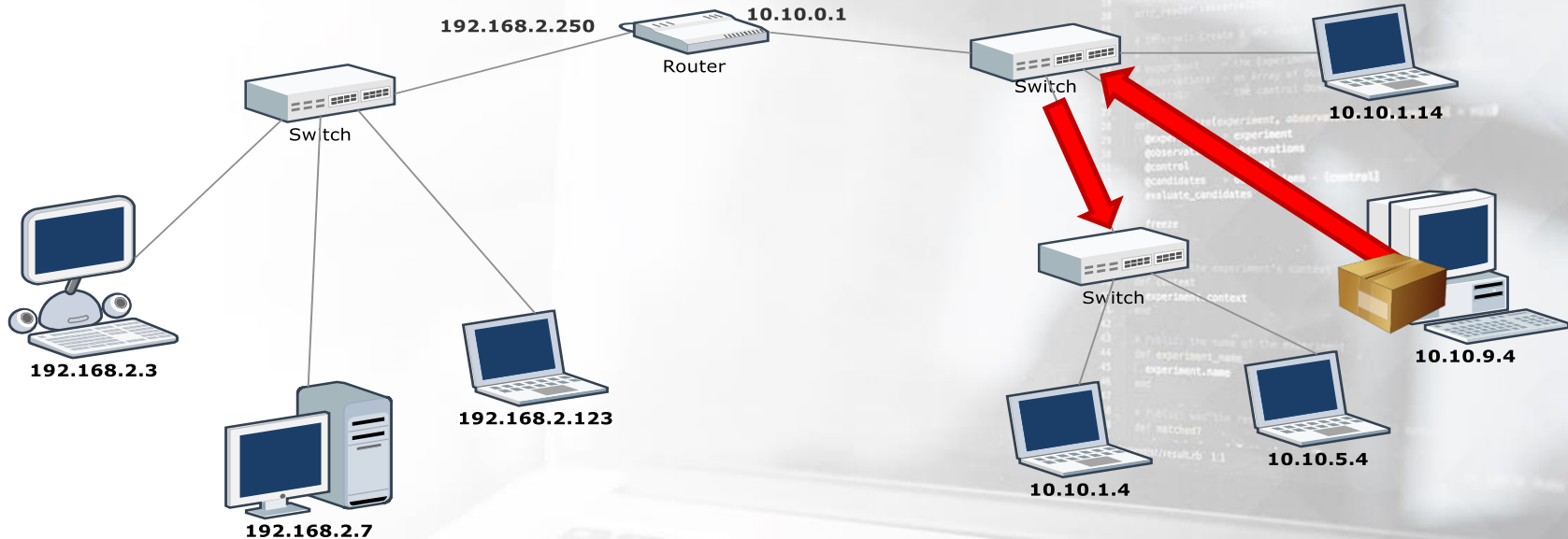
2.4.5.3 Multi-switch Example

What happens if 10.10.9.4 wants to send a packet to 10.10.1.4?



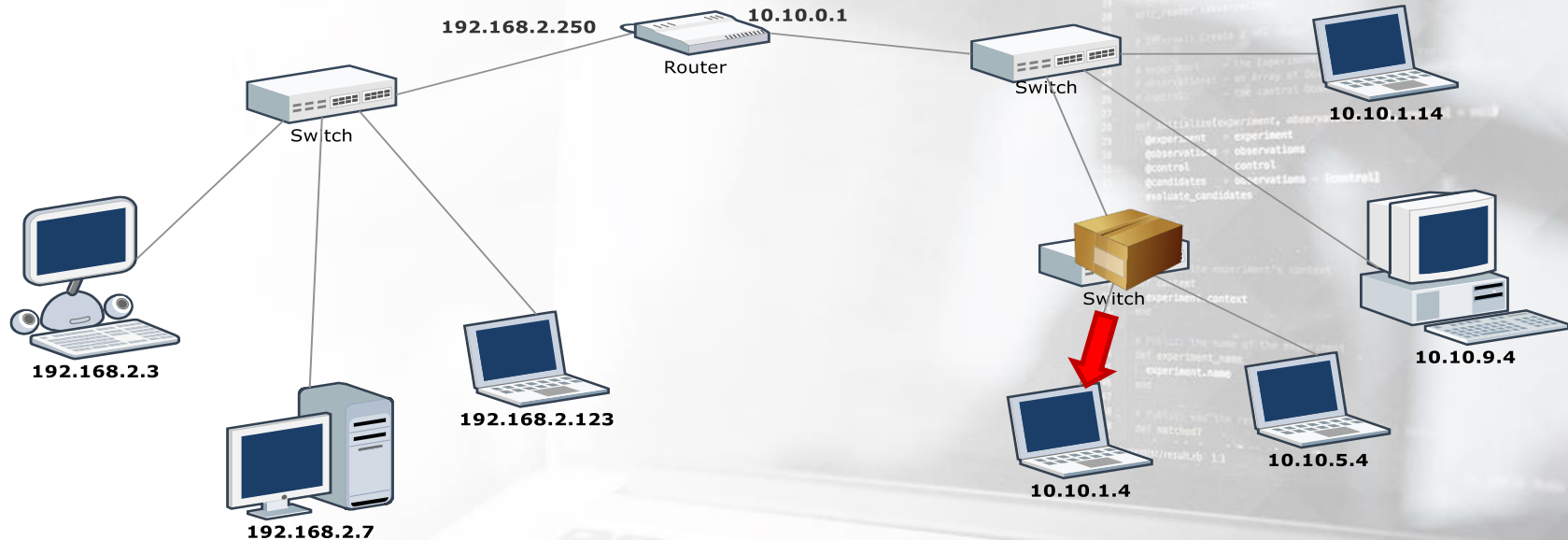
2.4.5.3 Multi-switch Example

The first switch receives the packet, performs a look-up in the CAM table and forwards it to the next switch.



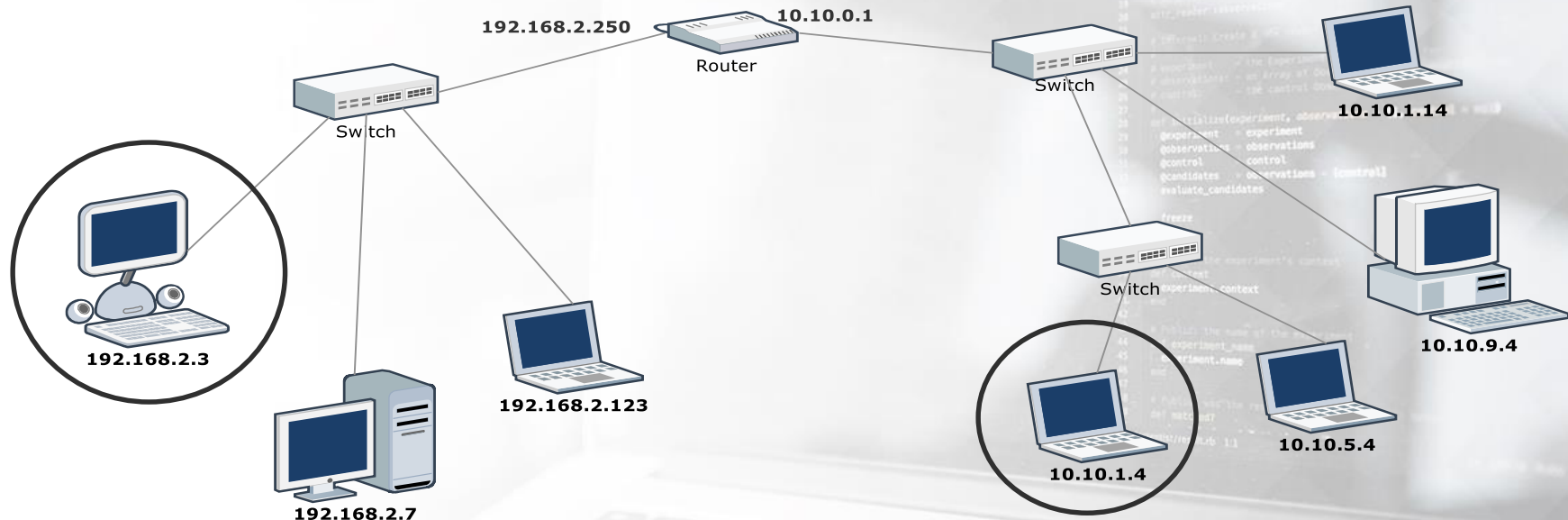
2.4.5.3 Multi-switch Example

The second switch forwards the packet to 10.10.1.4.



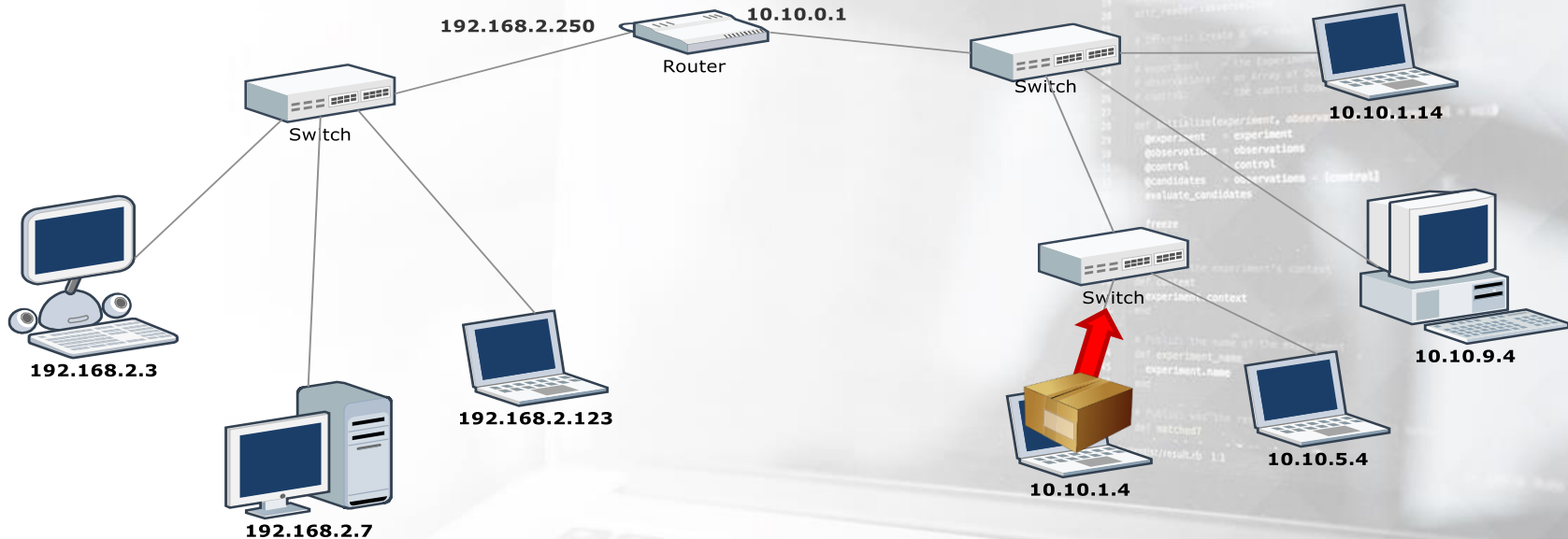
2.4.5.4 Multi-switch and Router Example

What happens if 10.10.1.4 wants to send a packet to 192.168.2.3?



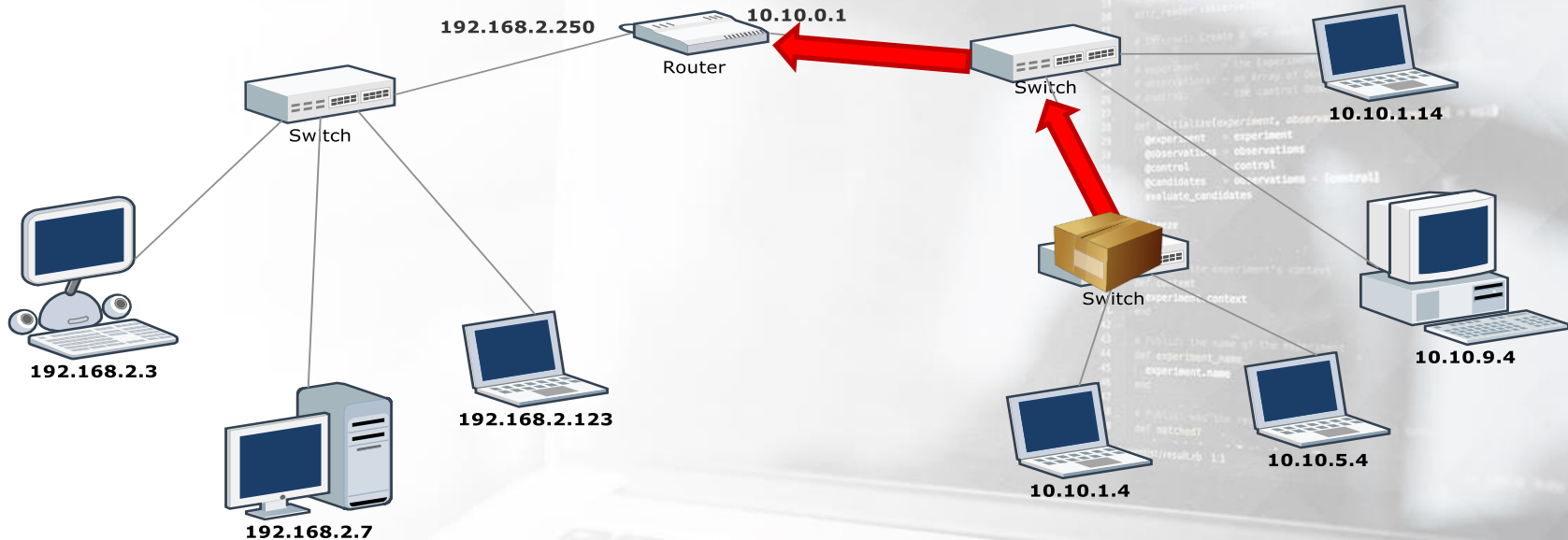
2.4.5.4 Multi-switch and Router Example

10.10.1.4 needs to send the packet to the router so that the first switch will forward the packet to the next one.



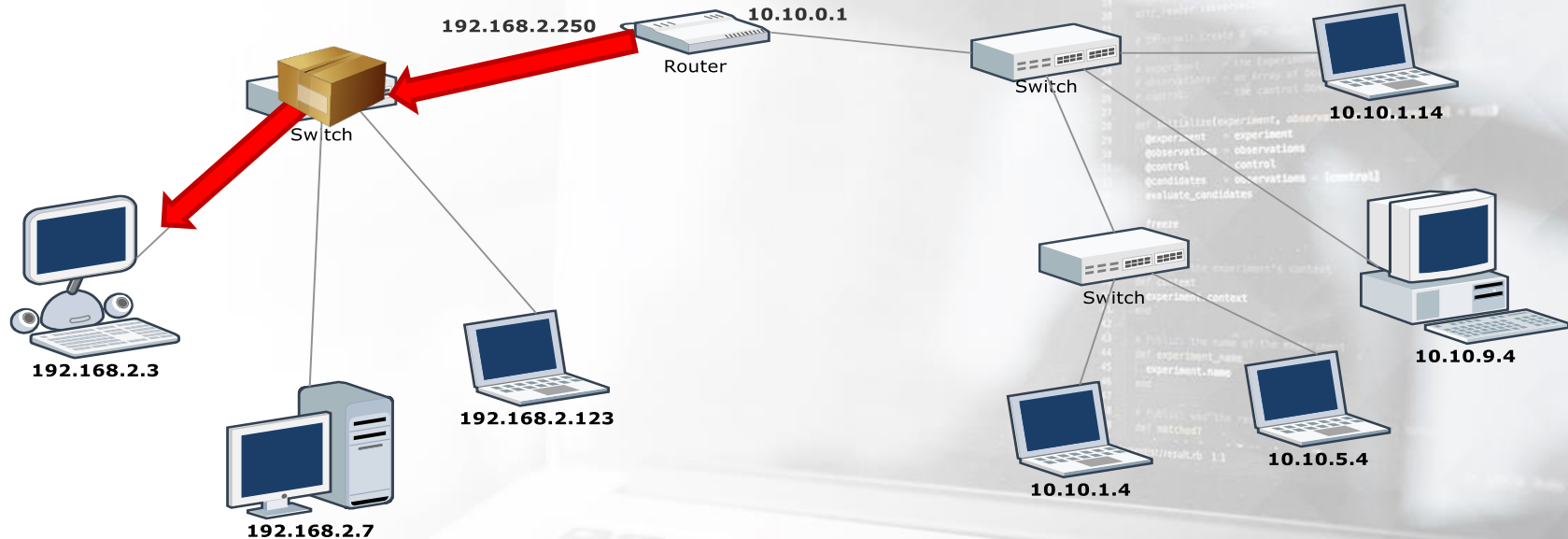
2.4.5.4 Multi-switch and Router Example

The packet then arrives at the router that, after a look up in the routing table, forwards it to the 192.168.2.0/24 network.



2.4.5.4 Multi-switch and Router Example

Finally, the packet is delivered.



2.4.5.5 Forwarding Tables

A forwarding table binds MAC addresses to interfaces.

In the following slides you will see:

- The structure of the table
- The way a switch constructs the MAC address – interface binding
- How forwarding works

```
24 def create_experiment()
25     # Create a new experiment
26     # Create a new experiment
27     # Create a new experiment
28     # Create a new experiment
29     # Create a new experiment
30     # Create a new experiment
31     # Create a new experiment
32     # Create a new experiment
33     # Create a new experiment
34     # Create a new experiment
35     # Create a new experiment
36     # Create a new experiment
37     # Create a new experiment
38     # Create a new experiment
39     # Create a new experiment
40     # Create a new experiment
41     # Create a new experiment
42     # Create a new experiment
43     # Create a new experiment
44     # Create a new experiment
45     # Create a new experiment
46     # Create a new experiment
47     # Create a new experiment
48     # Create a new experiment
49     # Create a new experiment
50     # Create a new experiment
51     # Create a new experiment
52     # Create a new experiment
53     # Create a new experiment
54     # Create a new experiment
55     # Create a new experiment
56     # Create a new experiment
57     # Create a new experiment
58     # Create a new experiment
59     # Create a new experiment
60     # Create a new experiment
61     # Create a new experiment
62     # Create a new experiment
63     # Create a new experiment
64     # Create a new experiment
65     # Create a new experiment
66     # Create a new experiment
67     # Create a new experiment
68     # Create a new experiment
69     # Create a new experiment
70     # Create a new experiment
71     # Create a new experiment
72     # Create a new experiment
73     # Create a new experiment
74     # Create a new experiment
75     # Create a new experiment
76     # Create a new experiment
77     # Create a new experiment
78     # Create a new experiment
79     # Create a new experiment
80     # Create a new experiment
81     # Create a new experiment
82     # Create a new experiment
83     # Create a new experiment
84     # Create a new experiment
85     # Create a new experiment
86     # Create a new experiment
87     # Create a new experiment
88     # Create a new experiment
89     # Create a new experiment
90     # Create a new experiment
91     # Create a new experiment
92     # Create a new experiment
93     # Create a new experiment
94     # Create a new experiment
95     # Create a new experiment
96     # Create a new experiment
97     # Create a new experiment
98     # Create a new experiment
99     # Create a new experiment
100    # Create a new experiment
```



2.4.5.5 Forwarding Tables

A typical forwarding table contains:

- The MAC address
- The interfaces the switch can use to deliver packets to a specific MAC address
- A time to live (TTL)

MAC	Interface	TTL
00:11:22:33:44:55	1	30
AA:BB:CC:DD:EE:01	2	5
AA:CC:FF:0A:0C:12	2	5
11:22:33:1D:CC:0A	3	7

2.4.5.5 Forwarding Tables

The forwarding table, or Content Addressable Memory table (CAM table), is stored in the device's RAM and is constantly refreshed with new information.

MAC	Interface	TTL
00:11:22:33:44:55	1	30
AA:BB:CC:DD:EE:01	2	5
AA:CC:FF:0A:0C:12	2	5
11:22:33:1D:CC:0A	3	7

2.4.5.5 Forwarding Tables

Looking at the table you can tell that:

- A single host is attached to Interface 1 and 3 respectively
- Two hosts are attached to interface 2 (probably via another switch).

MAC	Interface	TTL
00:11:22:33:44:55	1	30
AA:BB:CC:DD:EE:01	2	5
AA:CC:FF:0A:0C:12	2	5
11:22:33:1D:CC:0A	3	7

2.4.5.5 Forwarding Tables

There might be multiple hosts on the same interface and interfaces without any host attached.

In our example interface, 4 has no hosts attached.

MAC	Interface	TTL
00:11:22:33:44:55	1	30
AA:BB:CC:DD:EE:01	2	5
AA:CC:FF:0A:0C:12	2	5
11:22:33:1D:CC:0A	3	7

2.4.5.5 Forwarding Tables

The TTL determines how long an entry will stay in the table. This is important because the **CAM table has a finite size**.

So, as soon as an entry expires it is removed from the table.

MAC	Interface	TTL
00:11:22:33:44:55	1	30
AA:BB:CC:DD:EE:01	2	5
AA:CC:FF:0A:0C:12	2	5
11:22:33:1D:CC:0A	3	7

2.4.5.6 CAM Table Population

Switches learn new MAC addresses dynamically; they inspect the header of every packet they receive, thus identifying new hosts.

While routers use complex routing protocols to update their routing rules, switches just use the source MAC address of the packets they process to decide which interface to use when forwarding a packet.

2.4.5.6 CAM Table Population

The source MAC address is compared to the CAM table:

- If the MAC address is not in the table, the switch will add a new MAC-Interface binding to the table
- If the MAC-Interface binding is already in the table, its TTL gets updated
- If the MAC is in the table but bound to another interface the switch updates the table



2.4.5.7 Forwarding

To forward a packet:

- 1 The switch reads the destination MAC address of the frame.
- 2 It performs a look-up in the CAM table.
- 3 It forwards the packet to the corresponding interface.
- 4 If there is no entry with that MAC address, the switch will forward the frame to all its interfaces.

2.4.6 ARP

When a host wants to send a packet to another host, it needs to know the IP and the MAC address of the destination in order to build a proper packet.

You wouldn't be able to send your friend a letter if you don't know his/her address, right? What happens if the source host knows the IP address, but not the MAC address of the destination host?



2.4.6 ARP

This situation occurs in many circumstances, for example at every power up.

- A PC in an office knows a bunch of IP addresses, like the fileserver, the printers, and the webserver, but not their corresponding MAC addresses.

The host needs to know the MAC addresses of the other network nodes, and it can learn them by using the **Address Resolution Protocol (ARP)**.



2.4.6 ARP

With ARP a host can build the correct IP Address – MAC address binding.

This is one of the most fundamental protocols any modern network uses, so make sure to fully understand it.



2.4.6 ARP

When a host (*A*) wants to send traffic to another (*B*), and it only knows the IP address of *B*:

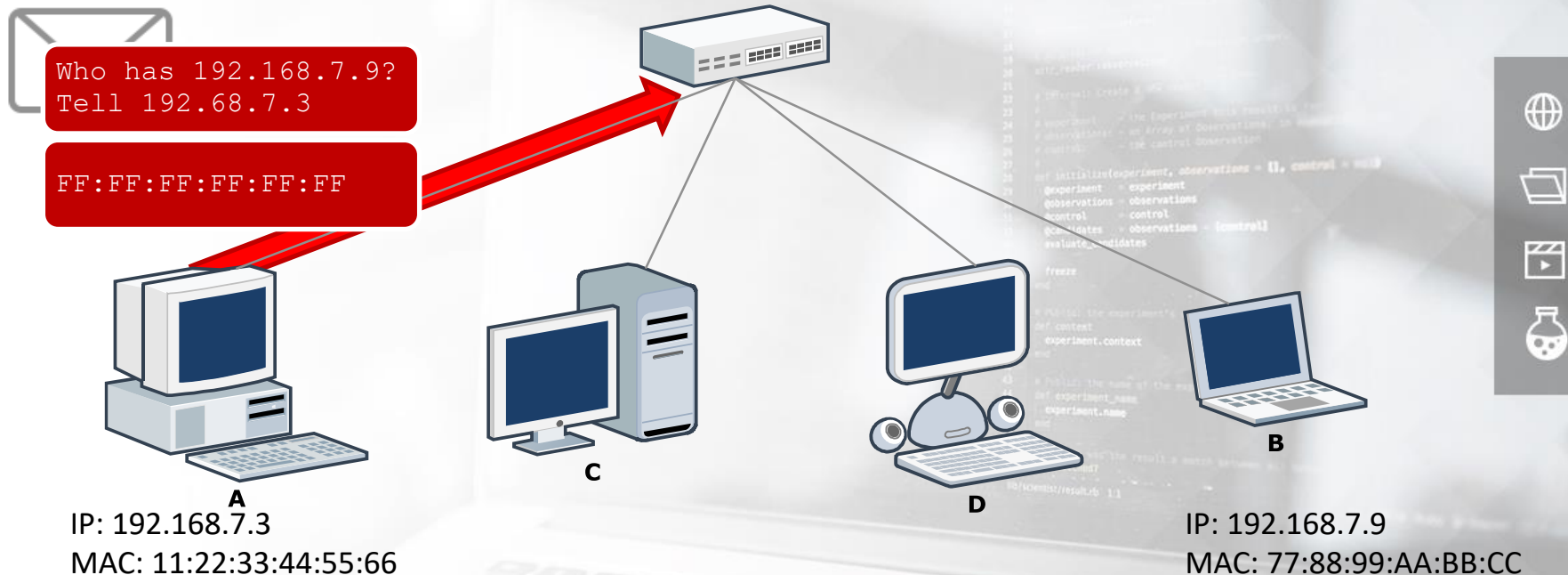
1. *A* builds an **ARP request** containing the IP address of *B* and FF:FF:FF:FF:FF:FF as destination MAC address. This is fundamental because the switches will forward the packet to every host.
2. Every host on the network will receive the request.
3. *B* replies with an **ARP reply**, telling *A* its MAC address.



2.4.6 ARP

1

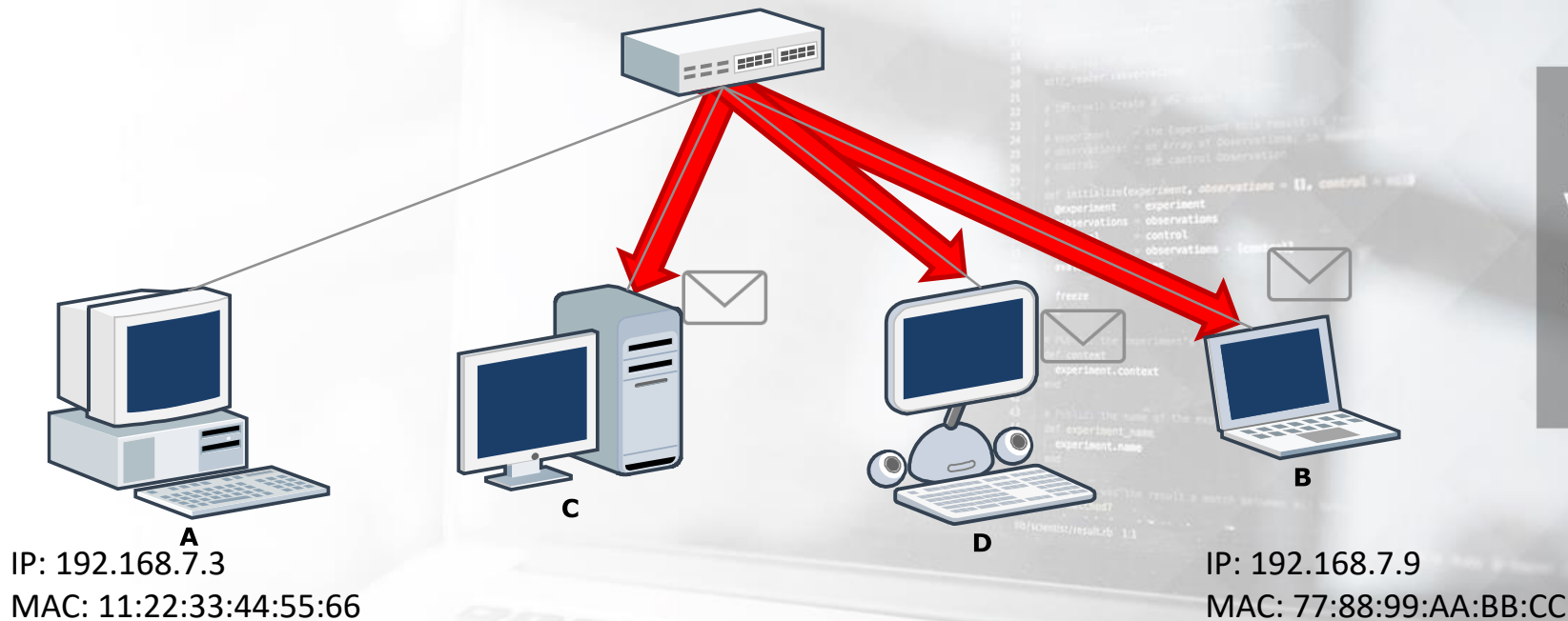
'A' sends a packet to the broadcast MAC address, asking for the MAC address of B.



2.4.6 ARP

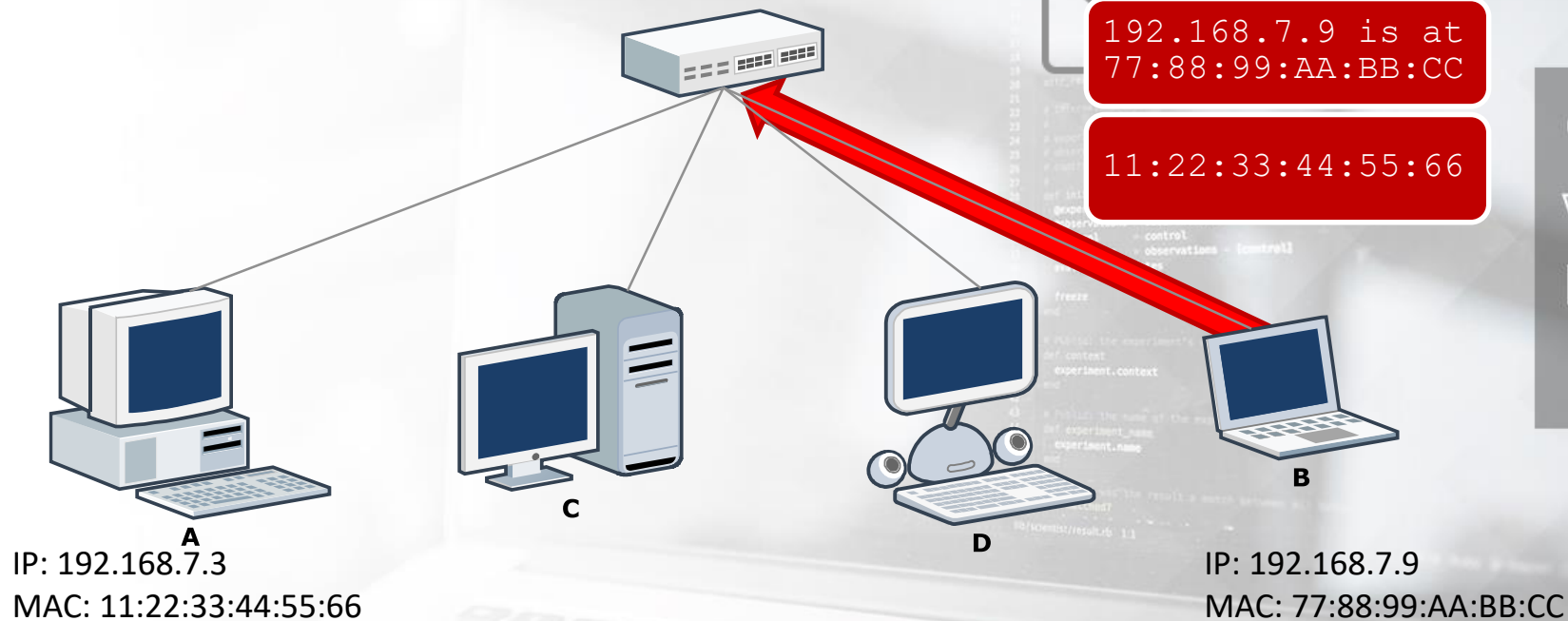
2

The switch forwards the packet to all its ports.



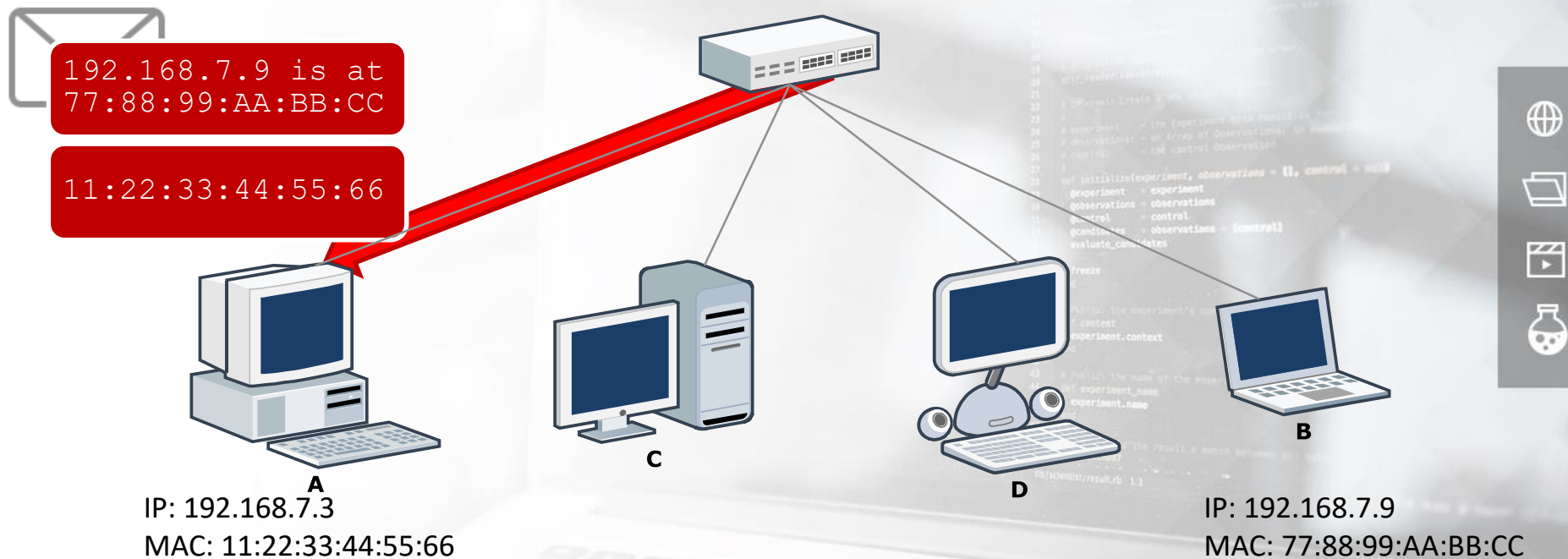
2.4.6 ARP

3 B replies telling A its MAC address.



2.4.6 ARP

Finally, the switch forwards the reply to A.



2.4.6 ARP

'A' will save the IP – MAC binding in its ARP cache. Further traffic to 'B' will not need a new ARP resolution protocol round.

ARP cache entries have a TTL too, as the size of the device RAM is finite. A host discards an entry at the power off or when the entry's TTL expires.

```
21 def control_loop(x):
22     # Create the experiment
23     # Create the experiment
24     # Create the experiment
25     # Create the experiment
26     # Create the experiment
27     # Create the experiment
28     # Create the experiment
29     # Create the experiment
30     # Create the experiment
31     # Create the experiment
32     # Create the experiment
33     # Create the experiment
34     # Create the experiment
35     # Create the experiment
36     # Create the experiment
37     # Create the experiment
38     # Create the experiment
39     # Create the experiment
40     # Create the experiment
41     # Create the experiment
42     # Create the experiment
43     # Create the experiment
44     # Create the experiment
45     # Create the experiment
46     # Create the experiment
47     # Create the experiment
48     # Create the experiment
49     # Create the experiment
50     # Create the experiment
51     # Create the experiment
52     # Create the experiment
53     # Create the experiment
54     # Create the experiment
55     # Create the experiment
56     # Create the experiment
57     # Create the experiment
58     # Create the experiment
59     # Create the experiment
60     # Create the experiment
61     # Create the experiment
62     # Create the experiment
63     # Create the experiment
64     # Create the experiment
65     # Create the experiment
66     # Create the experiment
67     # Create the experiment
68     # Create the experiment
69     # Create the experiment
70     # Create the experiment
71     # Create the experiment
72     # Create the experiment
73     # Create the experiment
74     # Create the experiment
75     # Create the experiment
76     # Create the experiment
77     # Create the experiment
78     # Create the experiment
79     # Create the experiment
80     # Create the experiment
81     # Create the experiment
82     # Create the experiment
83     # Create the experiment
84     # Create the experiment
85     # Create the experiment
86     # Create the experiment
87     # Create the experiment
88     # Create the experiment
89     # Create the experiment
90     # Create the experiment
91     # Create the experiment
92     # Create the experiment
93     # Create the experiment
94     # Create the experiment
95     # Create the experiment
96     # Create the experiment
97     # Create the experiment
98     # Create the experiment
99     # Create the experiment
100    # Create the experiment
```

2.4.6.1 Checking the ARP Cache

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

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



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



2.4.7 Hubs

Hubs were used in computer networks before switches. They have the same **purpose** but not the same **functionality**.

Hubs are simple repeaters that do not perform any kind of header check and simply forward packets by just repeating electric signals. They receive electric signals on a port and repeat the same signals on all the other ports.



2.4.7 Hubs

This means that every node on a hub-based network receives the same electric signals, thus the same packets.

Nowadays, hubs are very rare as they have mostly been replaced by switches.



TCP & UDP



2.5 TCP and UDP

How does this support my pentesting career?

- TCP Session Attacks
- Advanced DoS attacks
- Network scanning

2.5 TCP and UDP

Before checking out the different services that a transport layer protocol can offer to the application layer, let's consider something important about networks.

Computer networks can be **unreliable**. This means that some **packets can be lost** during their trip from source to destination. A packet can be lost because of network congestion, temporary loss of connection and other technical issues.



2.5 TCP and UDP

When designing a transport layer protocol, the designer must choose how to deal with these limitations. For example, TCP:

- **Guarantees packet delivery.** Because of that, an application that needs a guaranteed delivery will use TCP as the transport protocol.
- Is also **connection oriented**. It must establish a connection before transferring data.

Keep in mind these facts during your study!

```
def initialize(experiment, observations = [], candidates = [])
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - @control
  evaluate_candidates

  # Prepare the experiment's context
  def context
    experiment.context
  end

  # Fetch the name of the experiment
  def experiment_name
    experiment.name
  end

  # Check whether the results match between all
  def matches?
    # ...
  end
end
```



2.5 TCP and UDP

TCP is the most used transport protocol on the Internet. The vast majority of applications use it, and the IP protocol suite is often called **TCP/IP**.

Email clients, web browsers and FTP clients are some common applications using TCP.



2.5 TCP and UDP

On the other hand, UDP is much more simple than TCP:

- It does **not guarantee** packet delivery.
- It is **connectionless**.



2.5 TCP and UDP

UDP is faster than TCP, as it provides a **better throughput** (number of packets per second); in fact, it is used by **multimedia applications** that can tolerate packet loss but are throughput intensive.

For example, UDP is used for VoIP and video streaming: applications where you can tolerate a little glitch in the audio or video.



2.5 TCP and UDP

Here we can see a comparison table between TCP and UDP.

TCP	UDP
Lower throughput	Better throughput
Connection-oriented	Connectionless
Guarantees delivery	Does not guarantee packet delivery

2.5.1 Ports

Applications and their processes use TCP and UDP to send and receive data over the network. When an IP datagram reaches a host, how can the transport layer **know what the destination process is?**

We'll now introduce **ports**.

```
24 def experiment_results(self):
25     """Returns the experiment results as a list of observations"""
26     return self.observations
27
28 @property
29 def experiment(self):
30     """Returns the experiment object"""
31     return self._experiment
32
33 @property
34 def observations(self):
35     """Returns the observations object"""
36     return self._observations
37
38 @property
39 def control(self):
40     """Returns the control object"""
41     return self._control
42
43 @property
44 def candidates(self):
45     """Returns the candidates object"""
46     return self._candidates
47
48 def evaluate_candidates(self):
49     """Evaluates the candidates and returns the best one"""
50     # ...
51
52 def freeze(self):
53     """Freezes the experiment"""
54     # ...
55
56 def unfreeze(self):
57     """Unfreezes the experiment"""
58     # ...
59
60 def __str__(self):
61     """Returns a string representation of the experiment"""
62     return f"Experiment: {self.name}"
63
64 def __repr__(self):
65     """Returns a string representation of the experiment"""
66     return f"Experiment: {self.name}"
67
68 def __eq__(self, other):
69     """Returns True if the experiment is equal to the other experiment"""
70     return self.name == other.name
71
72 def __hash__(self):
73     """Returns the hash of the experiment"""
74     return hash(self.name)
75
76 def __getitem__(self, key):
77     """Returns the value of the experiment"""
78     return self.__dict__[key]
79
80 def __setitem__(self, key, value):
81     """Sets the value of the experiment"""
82     self.__dict__[key] = value
83
84 def __delitem__(self, key):
85     """Deletes the value of the experiment"""
86     del self.__dict__[key]
87
88 def __len__(self):
89     """Returns the length of the experiment"""
90     return len(self.__dict__)
91
92 def __iter__(self):
93     """Returns an iterator over the experiment"""
94     return iter(self.__dict__)
95
96 def __contains__(self, key):
97     """Returns True if the key is in the experiment"""
98     return key in self.__dict__
99
100 def __getattr__(self, name):
101     """Returns the attribute of the experiment"""
102     return self.__dict__[name]
```

2.5.1 Ports

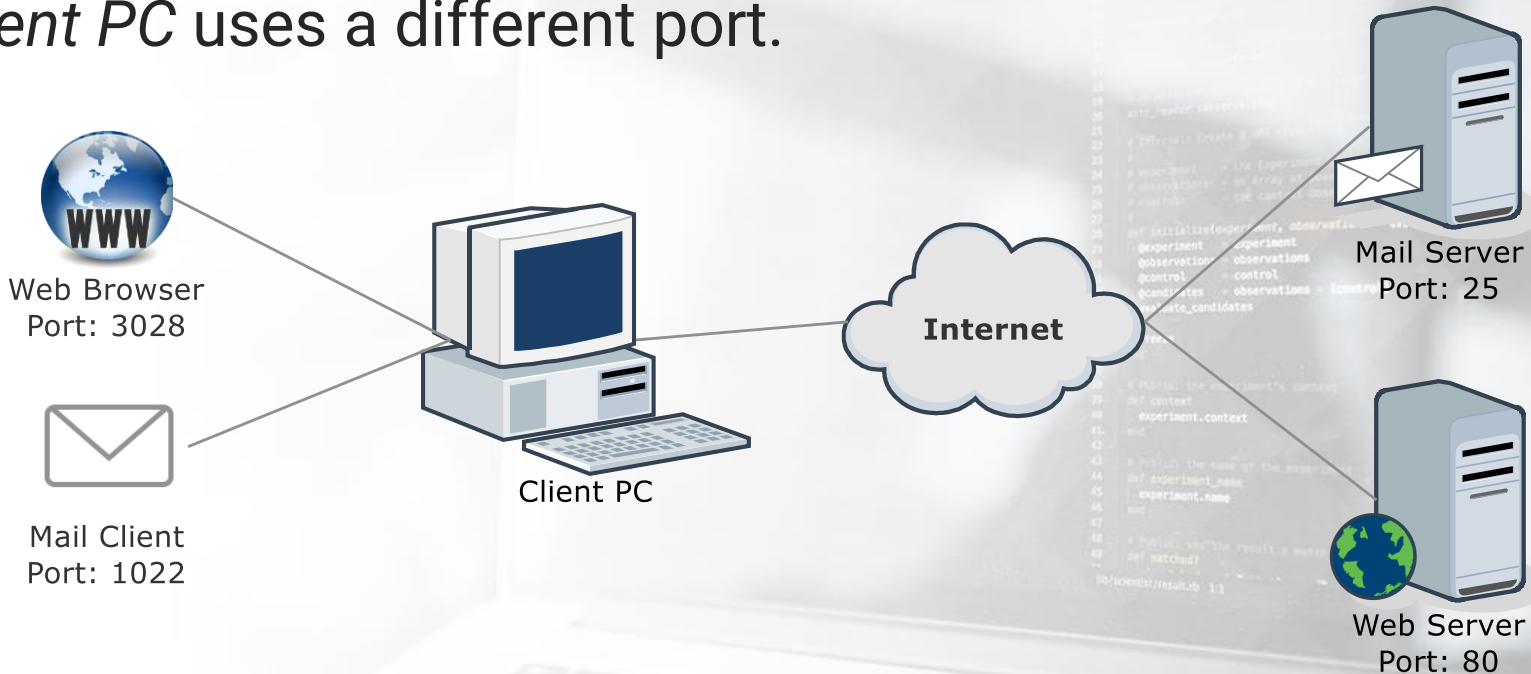
Ports are used to identify a single network **process** on a machine. If you want to unequivocally identify a process on a network, you need to know the **<IP>:<Port> pair**.

As an example, you can compare the port to the recipient's name on a letter; the street address (IP) identifies the building, while the person name identifies the final recipient of the letter.



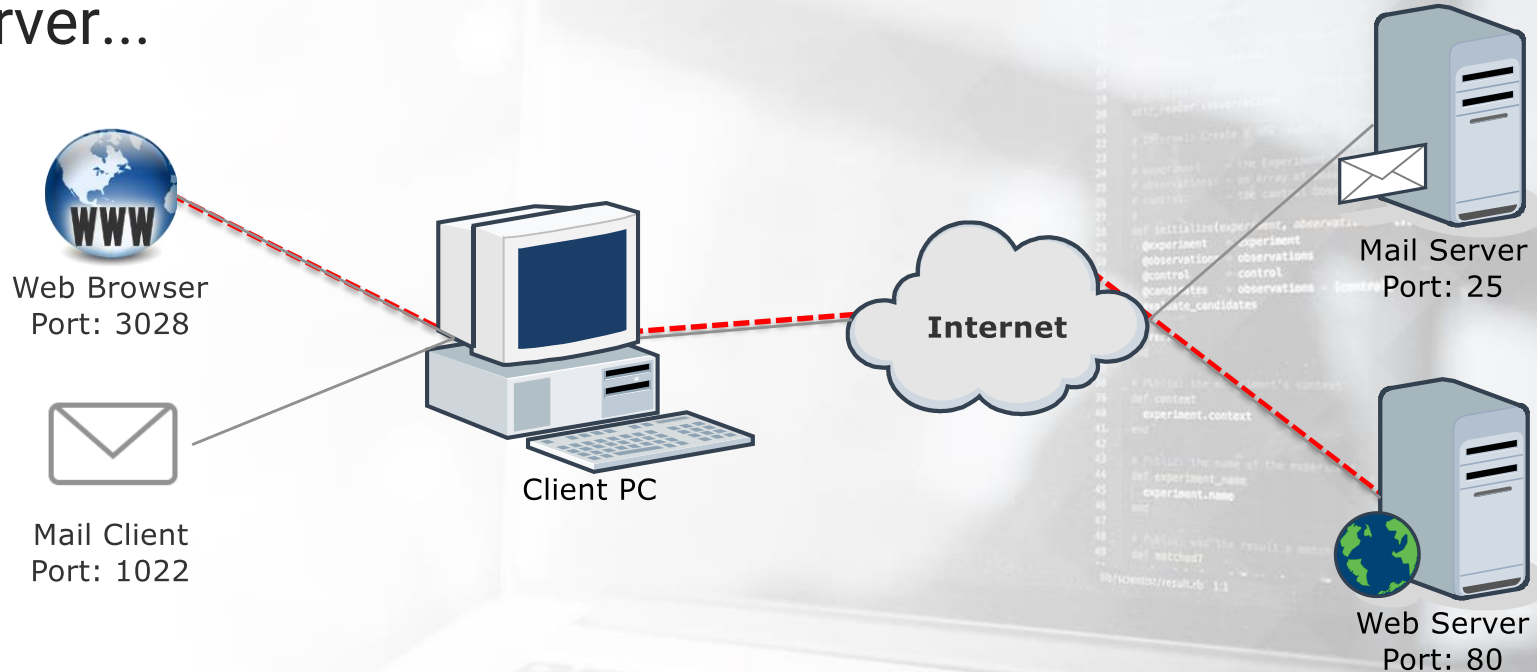
2.5.1.1 Ports Examples

In this image, you can see how every client application on *Client PC* uses a different port.



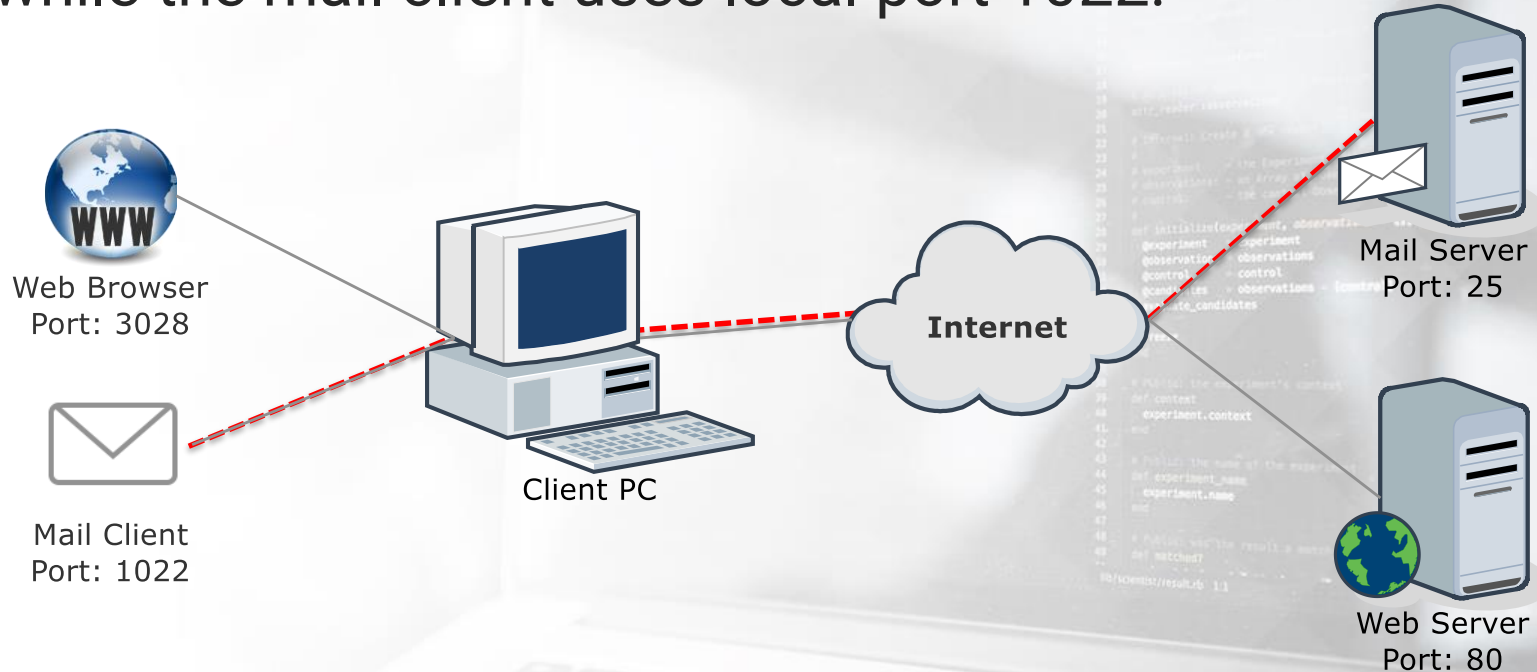
2.5.1.1 Ports Examples

The browser uses local port 3028 to connect to the web server...



2.5.1.1 Ports Examples

... while the mail client uses local port 1022.



2.5.1.1 Ports Examples

In the previous example:

- All the communication from the web browser to the web server will have 3028 as the source port and 80 as the destination port.
- All the communication back from the web server to the browser will have 80 as the source port and 3028 as the destination port.

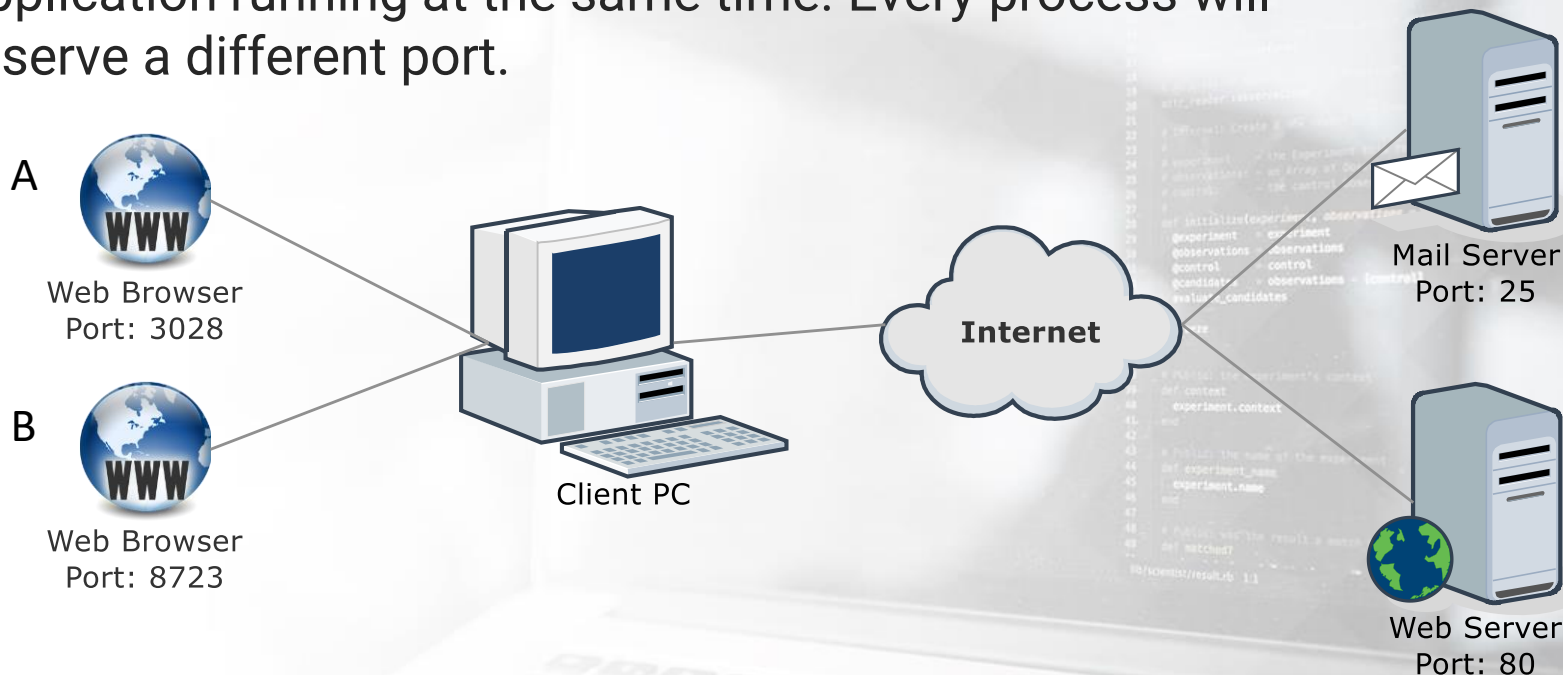
2.5.1.1 Ports Examples

Similarly, for the mail client and server:

- All the communication from the mail client to the server will have 1022 as the source port and 25 as the destination port.
- All the communication back from the mail server to the mail client will have 25 as the source port and 1022 as the destination port.

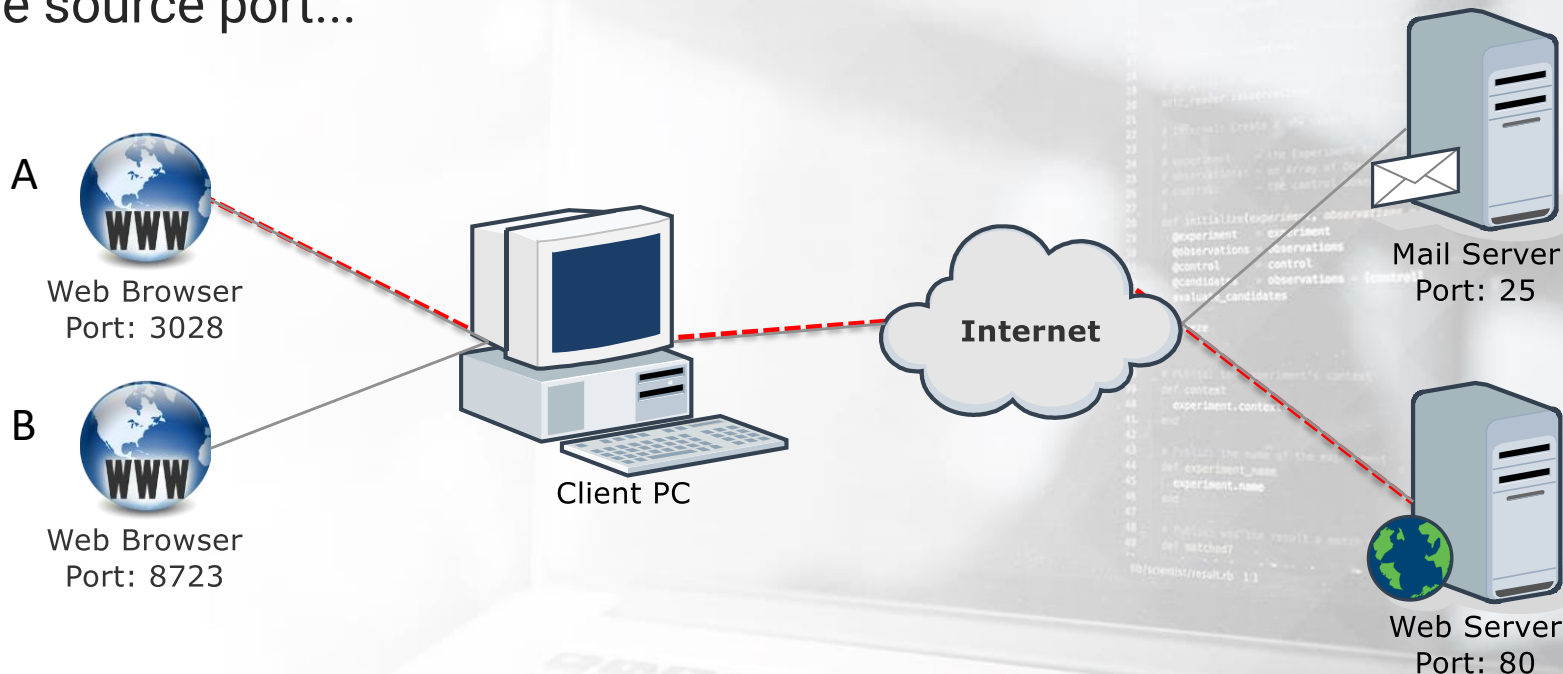
2.5.1.1 Ports Examples

Furthermore, you may also have multiple instances of the same application running at the same time. Every process will reserve a different port.



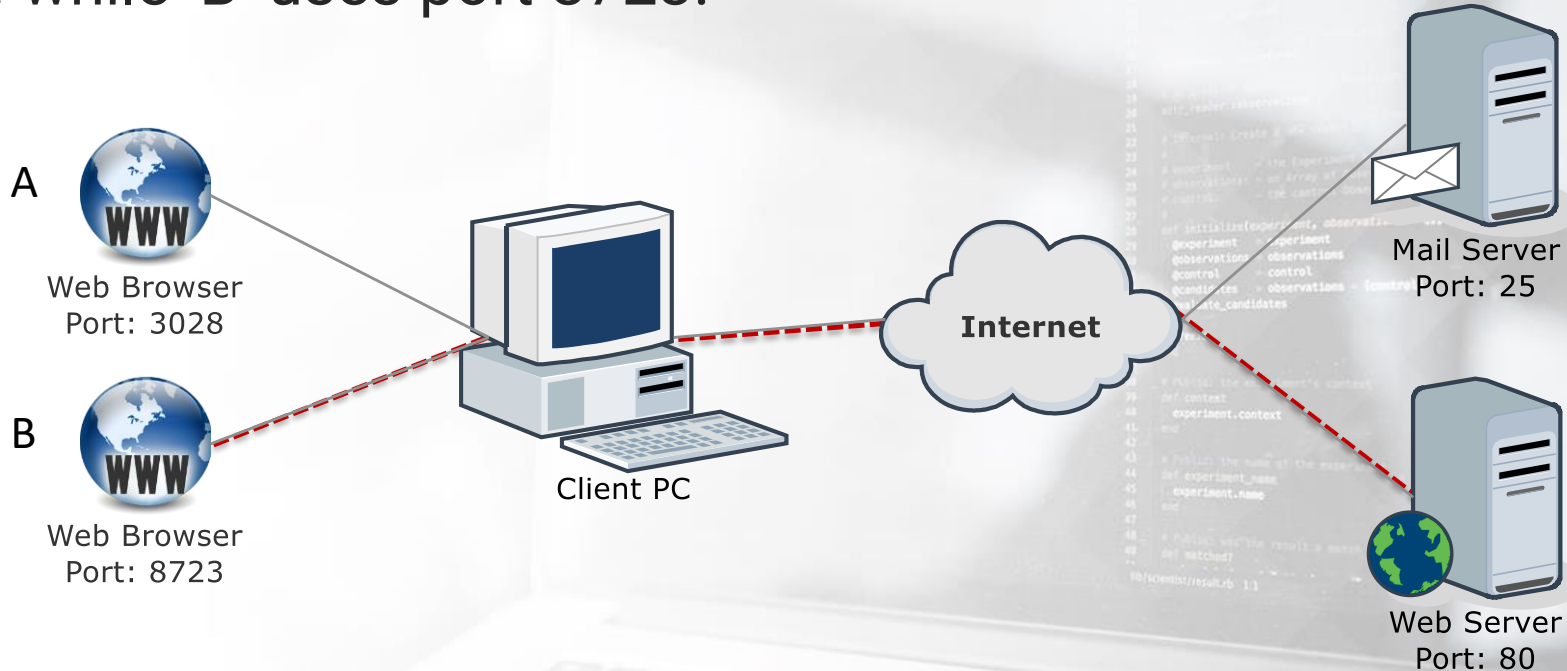
2.5.1.1 Ports Examples

In this example, 'A' communicates with the web server using 3028 as the source port...



2.5.1.1 Ports Examples

... while 'B' uses port 8723.



2.5.1 Ports

To correctly address a process on a network, you have to refer to the `<IP>:<Port>` pair. For example:

- 192.168.5.3:80
- 10.11.12.1:443
- 172.16.8.9:22

But, how can you know the right port for a common service?



2.5.2 Well-known Ports

Ports in the ranging from **0-1023**, the first 1024 that is, are called **well-known ports** and are used by servers for the most common services.

For example, when a web browser connects to a server via HTTPS, the user does not have to manually specify 443 as the destination port.



2.5.2 Well-known Ports

Each common protocol has a well-known port in the 0-1023 range. Common server processes, or daemons, use well-known ports most of the time.

Ports are assigned by IANA and are referenced in [this document](#).

```
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

2.5.2 Well-known Ports

You do not need to know all the service port assignments, but you should at least remember the most common, such as:

- SMTP (25)
- SSH (22)
- POP3 (110)
- IMAP (143)
- HTTP (80)
- HTTPS (443)
- NETBIOS (137, 138, 139)
- SFTP (115)
- Telnet (23)
- FTP (21)
- RDP (3389)
- MySQL (3306)
- MS SQL Server (1433)



2.5.2 Well-known Ports

As briefly introduced before, a **daemon** is a program that runs a service. System administrators can change the daemon configuration, **changing the port** the service listens to for connection. They do that to make services recognition a little bit harder for hackers.

For example, you could find an FTP daemon listening on port 4982 instead of 21 or SSH listening on port 8821.



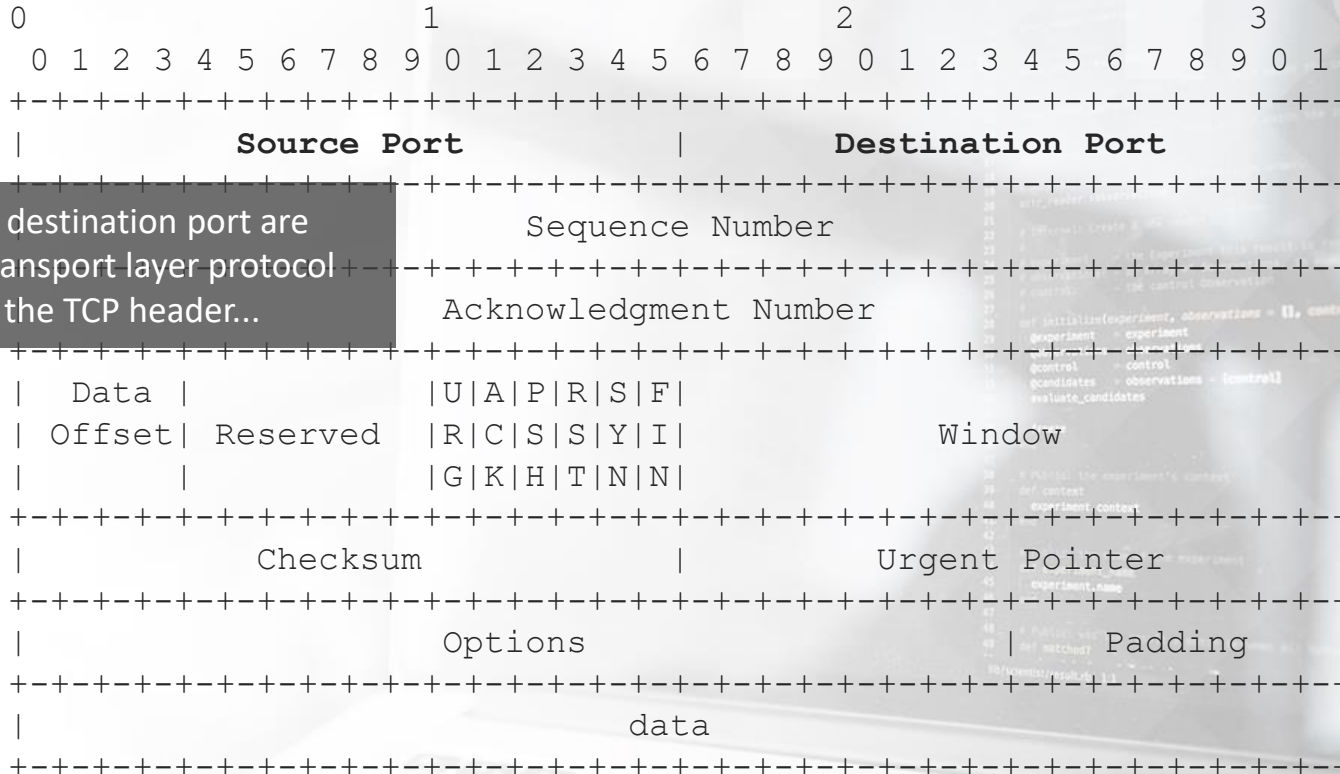
2.5.3 TCP and UDP headers

Let's now see how ports are used by applications.

How can server and client applications know which port to use? They use two fields in the TCP or UDP header: the **source** and **destination** ports.

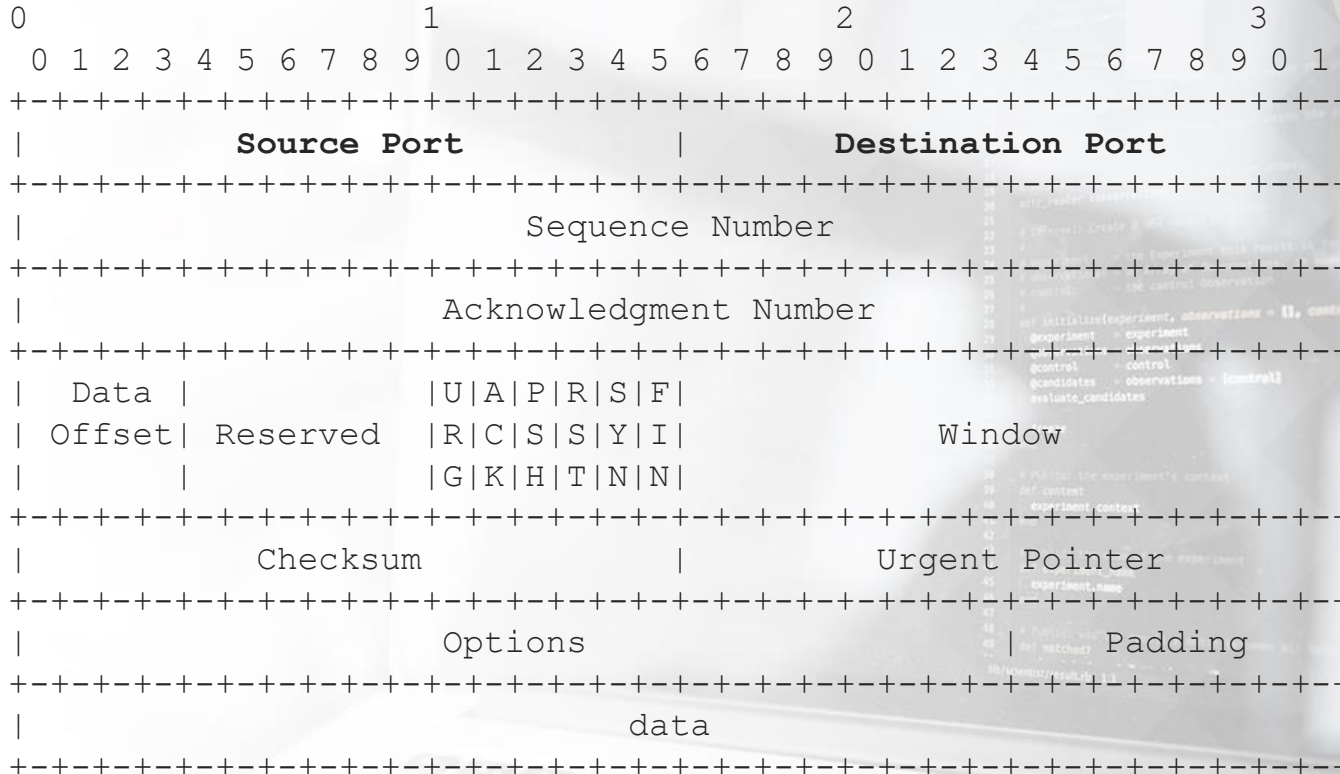


2.5.3.1 TCP Header



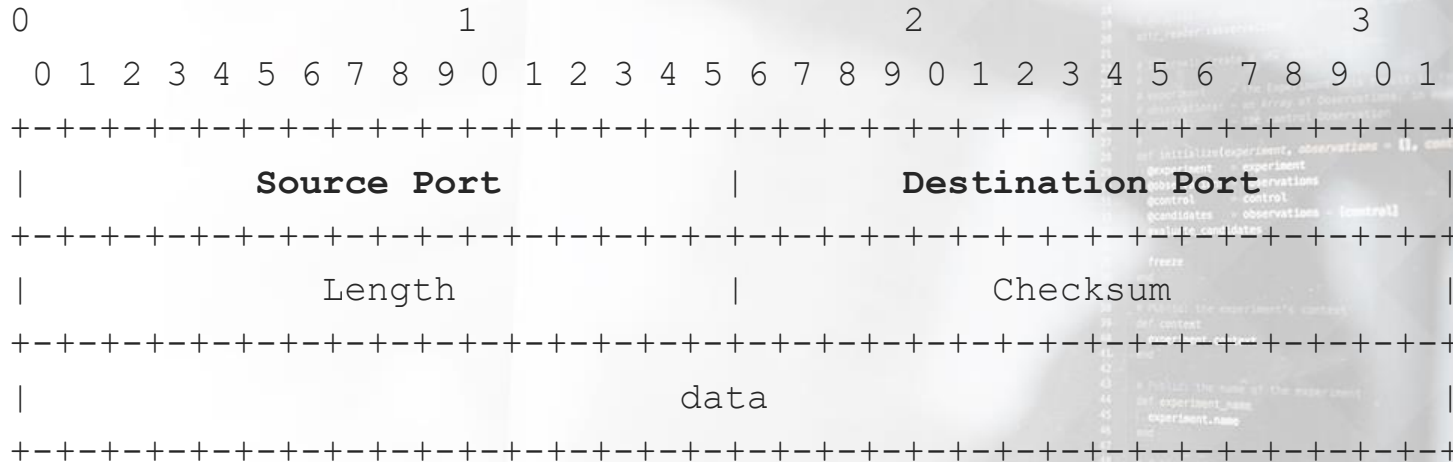
The source and destination port are included in the transport layer protocol header. Like the TCP header...

2.5.3.1 TCP Header

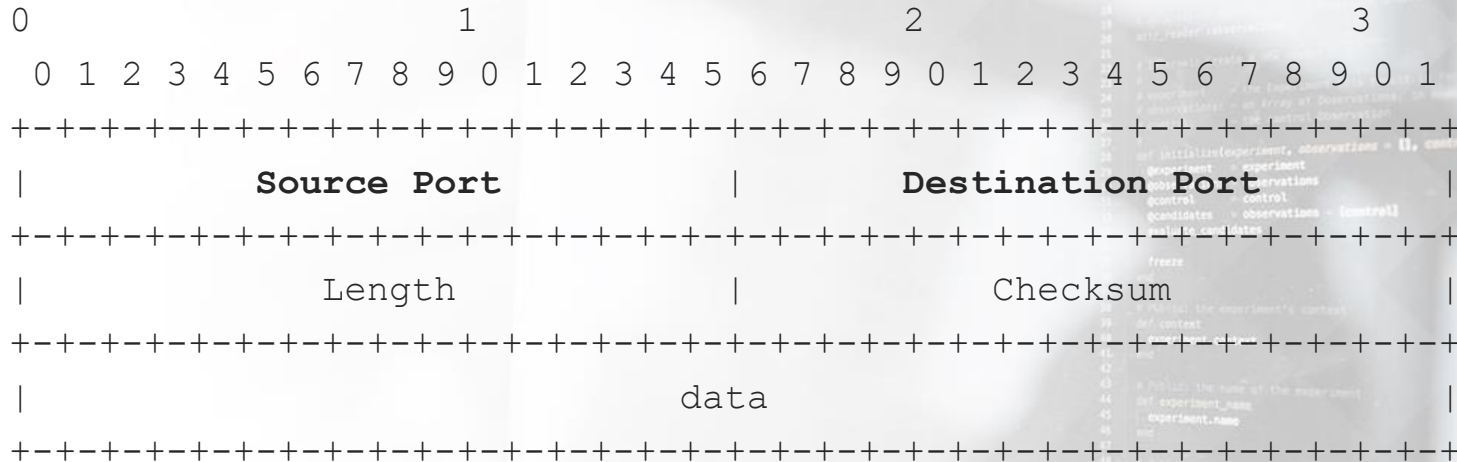


2.5.3.2 UDP Header

...or the UDP header.



2.5.3.2 UDP Header



2.5.4 Netstat Command

To check the listening ports and the current (TCP) connections on a host you can use:

- `netstat -ano` on Windows
- `netstat -tunp` on Linux
- `netstat -p tcp -p udp` together with
`lsof -n -i4TCP -i4UDP` on MacOS

Use these commands to show information about the processes listening on the machine and processes connecting to remote servers.

2.5.4 Netstat Command

Another great tool for Windows is TCPView from Sysinternals.

TCPView shows:

- Process name
- PID
- Protocol
- Local and remote addresses
- Local and remote ports
- State of the connection (if applicable)

2.5.5 TCP Three Way Handshake

We have seen that TCP is **connection oriented**. Now, let's look at how TCP connections work, as well as highlight the most important factors involved, from the penetration tester's point of view, in a 3-way handshake.

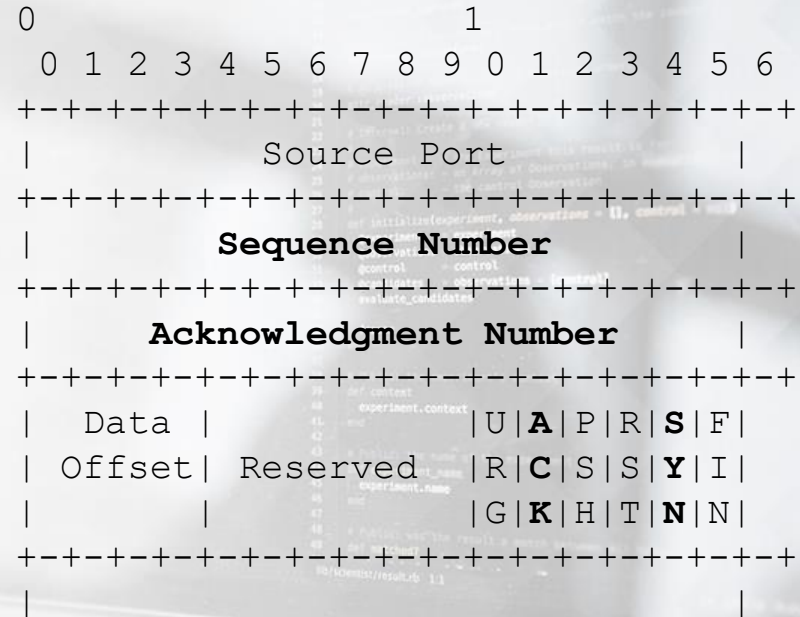
To establish a connection between two hosts running TCP, they must perform three steps: the **three-way handshake**. They can then start the actual data transmission.



2.5.5 TCP Three Way Handshake

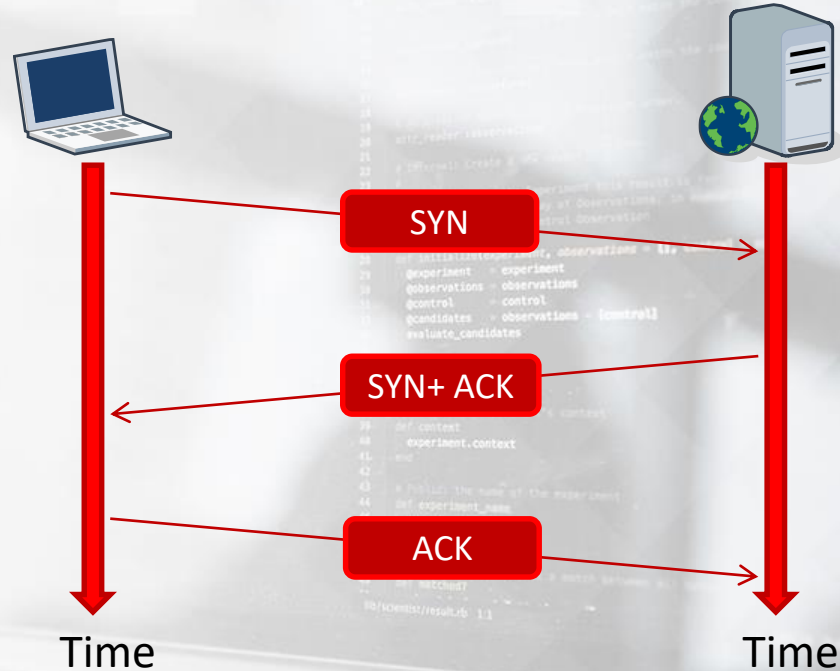
The header fields involved in the handshake are:

- Sequence number
- Acknowledgement numbers
- SYN and ACK flags



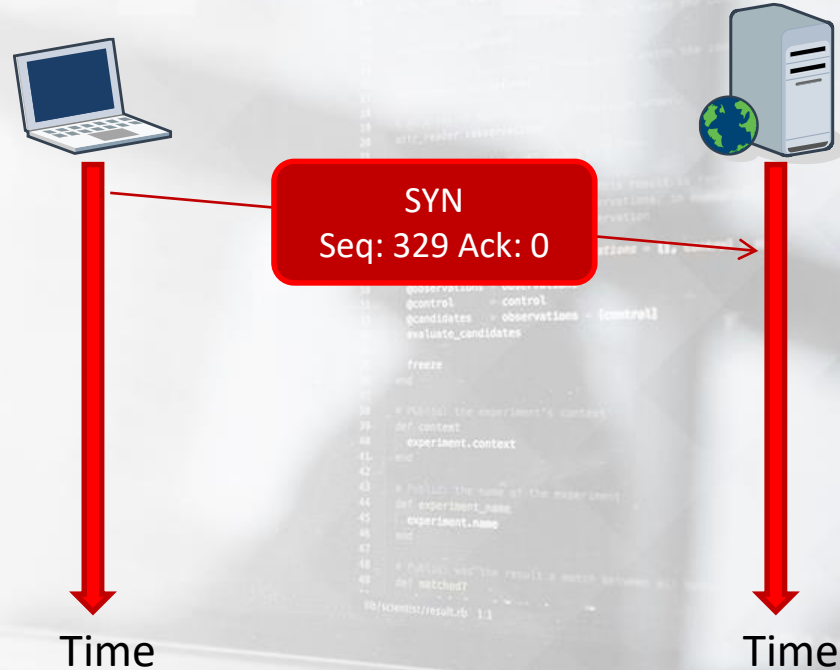
2.5.5 TCP Three Way Handshake

The steps in the handshake are used to synchronize the sequence and acknowledgment numbers between the server and the client.



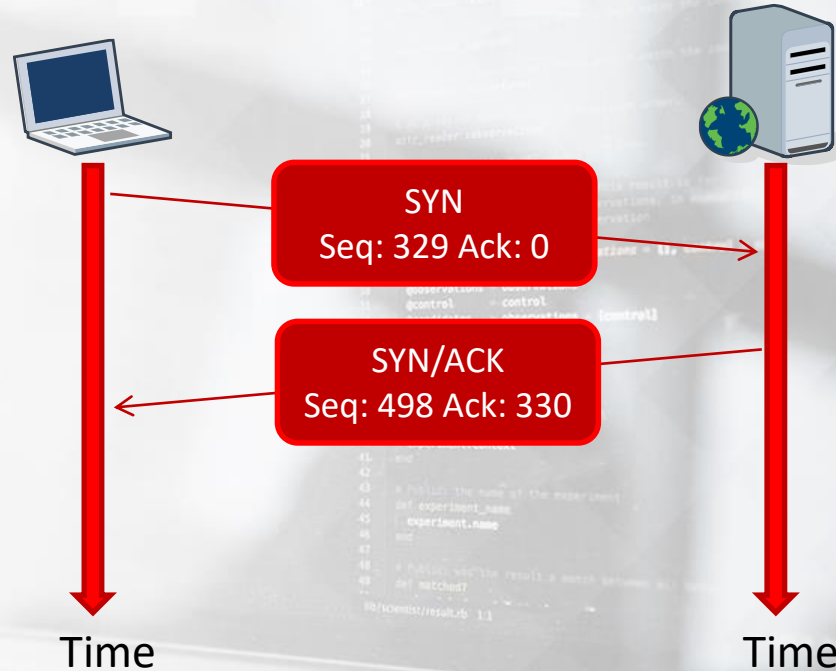
2.5.5 TCP Three Way Handshake

During the first step, the client sends a TCP packet to the server with the SYN flag enabled and a random sequence number.



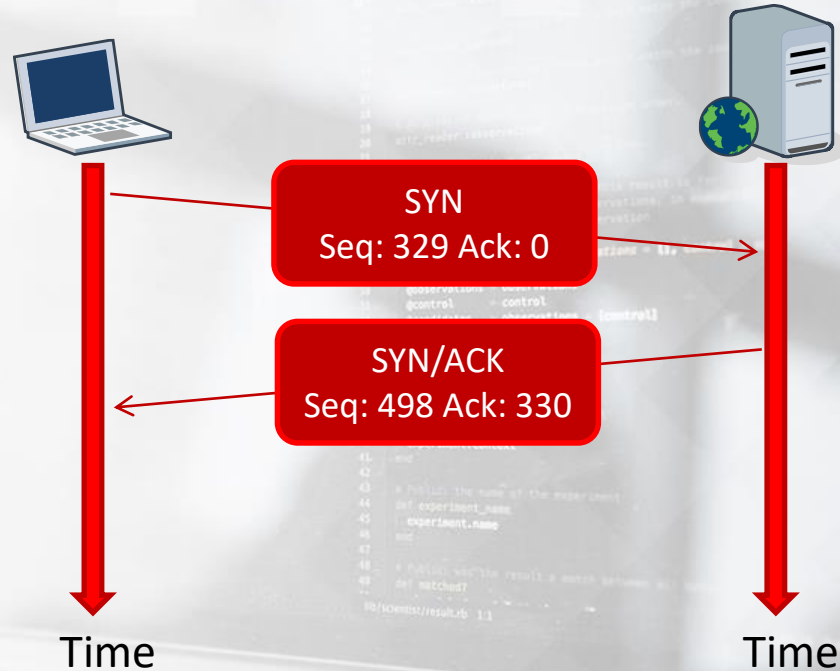
2.5.5 TCP Three Way Handshake

In the second step, the server replies by sending a packet with both the SYN and ACK flag set and another random sequence number.



2.5.5 TCP Three Way Handshake

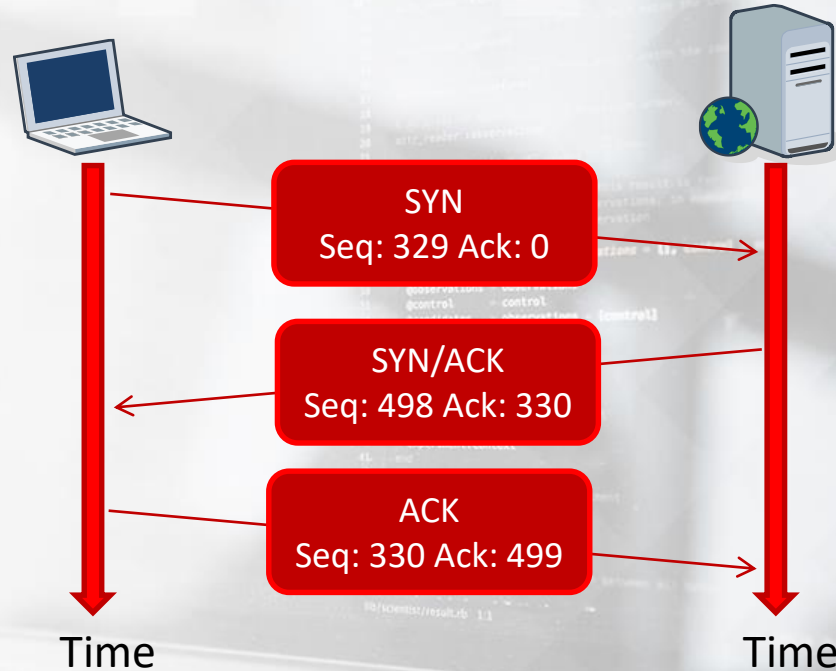
The ACK number is always a simple increment of the SYN number sent by the client.



2.5.5 TCP Three Way Handshake

Finally, the client completes the synchronization by sending an ACK packet.

Note that the client behaves just like the server when sending ACK packets.



2.5.6 References

For additional information, please check out the links below:

- IP Layer Network Administration with Linux
- TCP/IP Tutorial and Technical Overview.
- Packet Analysis Reference Guide v3.0.

**To access the above attachments, go to the course in your members area and click the resources drop-down in the appropriate module line.*

Firewalls and Network Defense



2.6 Firewalls and Network Defense

There are many different appliances on the market that a system administrator can use to protect the network.

These devices use **different techniques** and work on **different layers** to perform **access control, attack detection and prevention**.



2.6 Firewalls and Network Defense

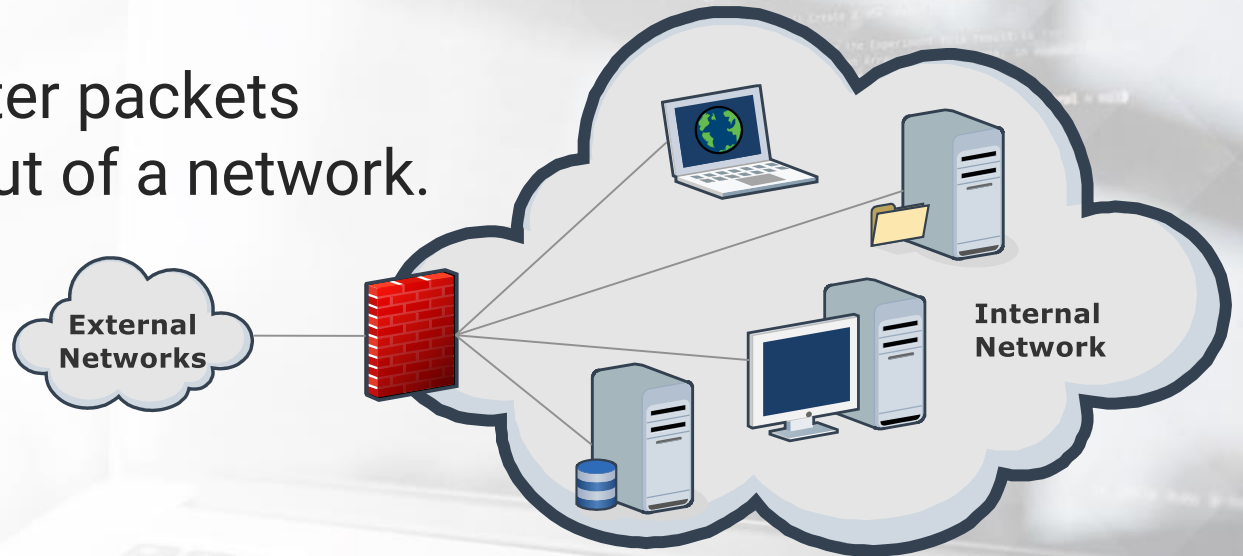
How does this support my pentesting career?

- Evading firewalls
- Advanced stealth scanning
- Filtering evasion

2.6.1 Firewalls

Firewalls are specialized software modules running on a computer or a dedicated network device.

They serve to filter packets coming in and out of a network.



2.6.1 Firewalls

Firewalls help system administrators and desktop users to control the access to network resources and services.

A firewall can **work on different layers** of the OSI model, thus providing different features and protections.



2.6.1 Firewalls

It is imperative that you understand how firewalls work and what kind of threats they prevent.

Many people believe that firewalls and antiviruses are all they need to stay secure, in the following slides you will see why this idea is wrong.



2.6.2 Packet Filtering Firewalls

The most basic feature of a firewall is **packet filtering**.

With packet filtering, an administrator can create rules which will filter packets according to certain characteristics like:

- Source IP address
- Destination IP address
- Protocol
- Source port
- Destination Port



2.6.2 Packet Filtering Firewalls

Packet filters run on home DSL routers as well as high-end enterprise routers and are the cornerstone of network defense.



2.6.2 Packet Filtering Firewalls

Packet filters inspect the header of every packet to choose how to treat the packet. The more common actions are:

- **Allow:** allow the packet to pass
- **Drop:** drops the packet without any diagnostic message to the packet source host
- **Deny:** do not let the packet pass, but notify the source host

2.6.2 Packet Filtering Firewalls

Inspecting the header of a packet does not give you any information on the **actual packet content**.

In a company, network administrators configure a corporate firewall to allow web browsing from the internal network. They do that by allowing TCP traffic to have 80 or 443 as destination ports. What happens if an internal machine tries to connect via SSH to a server listening on port 80?

The traffic will pass even if it is not web traffic!



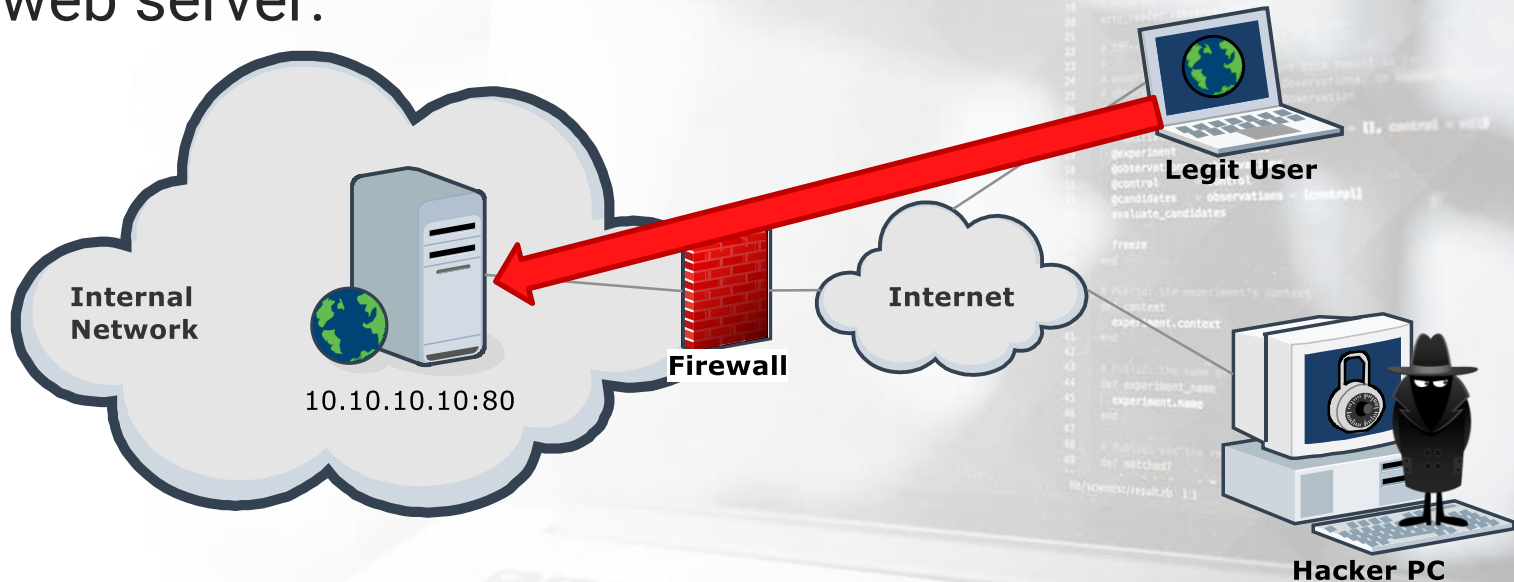
2.6.2.1 Packet Filtering vs. Application Attacks

Let's now look at a typical example where packet inspection does not stop a hacker from exploiting a server.



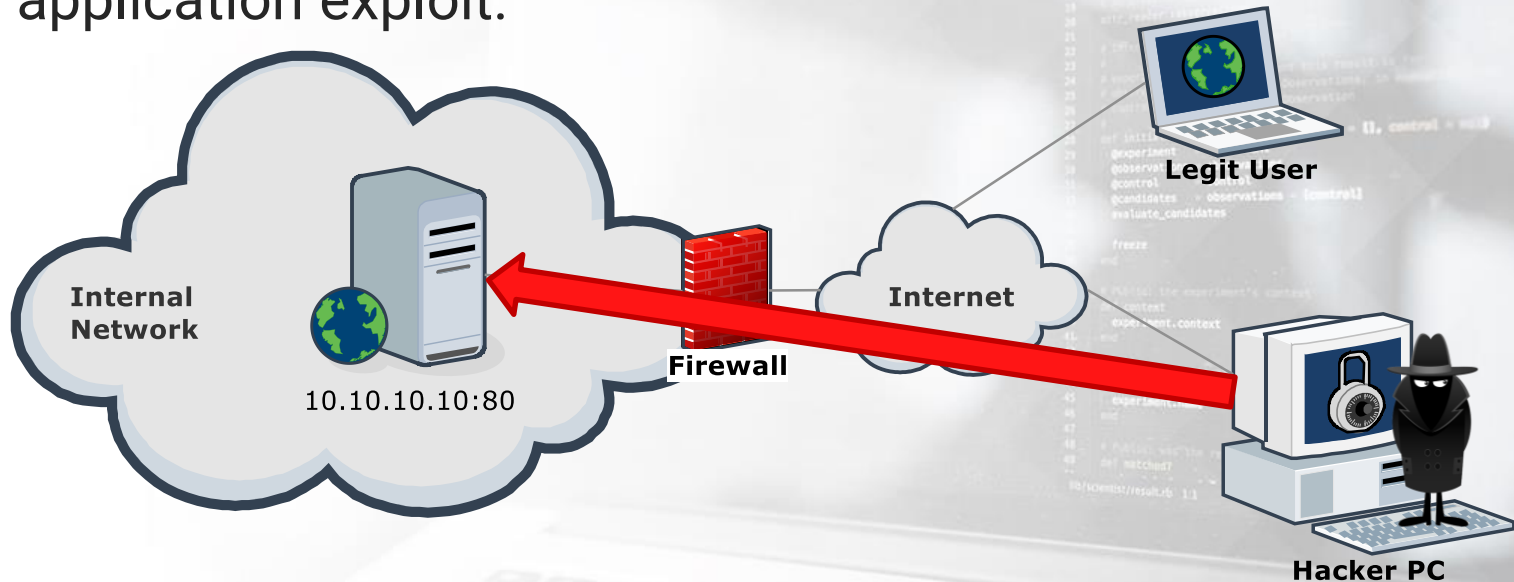
2.6.2.1 Packet Filtering vs. Application Attacks

A company hosts a web server. The firewall will allow all incoming traffic from the Internet and direct it to port 80 of the web server.



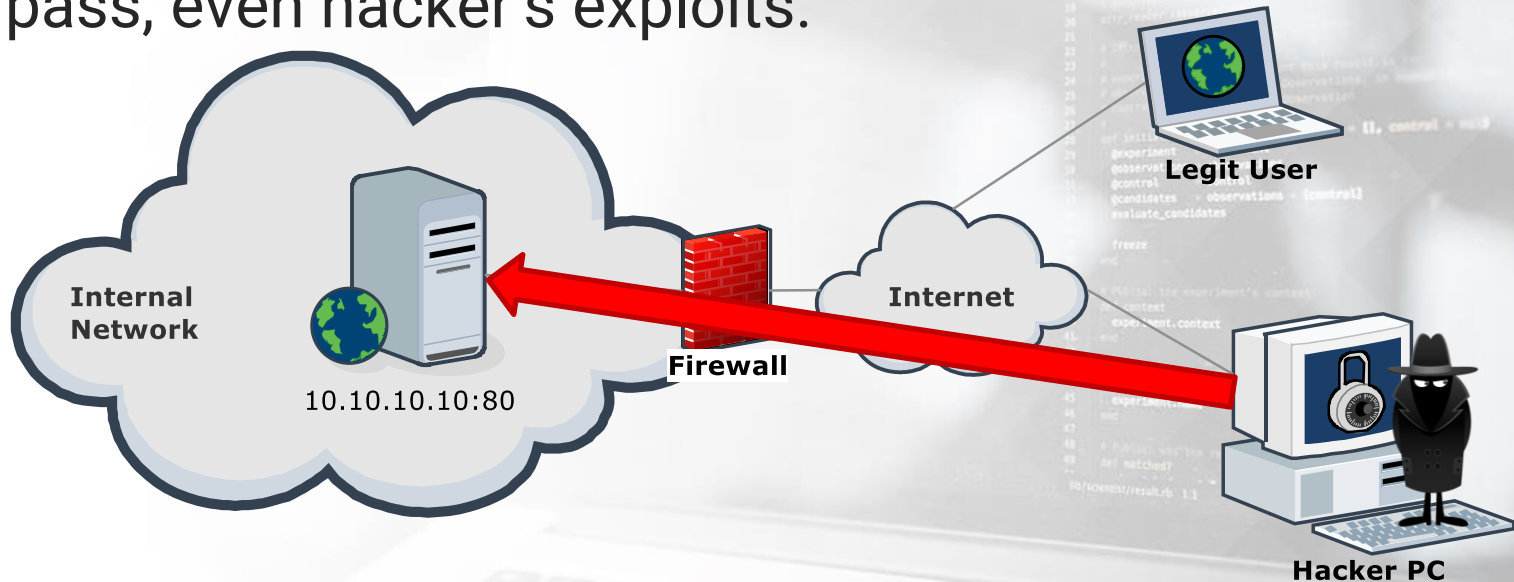
2.6.2.1 Packet Filtering vs. Application Attacks

In the same way, application exploits will go through, because the firewall cannot see the difference between web browsing and a web application exploit.



2.6.2.1 Packet Filtering vs. Application Attacks

The firewall can only filter traffic by using IP addresses, ports, and protocols. **Any** kind of application layer traffic will pass, even hacker's exploits.



2.6.2.2 Packet Filtering vs. Trojan Horse

A hacker manages to exploit a workstation in a company network and wants to install a Trojan horse to remotely manage that machine.

The Trojan, by default, can be configured to **accept** connections on port 123 TCP or UDP or to **connect back** to the hacker's machine on port 123 TCP or UDP.



2.6.2.2 Packet Filtering vs. Trojan Horse

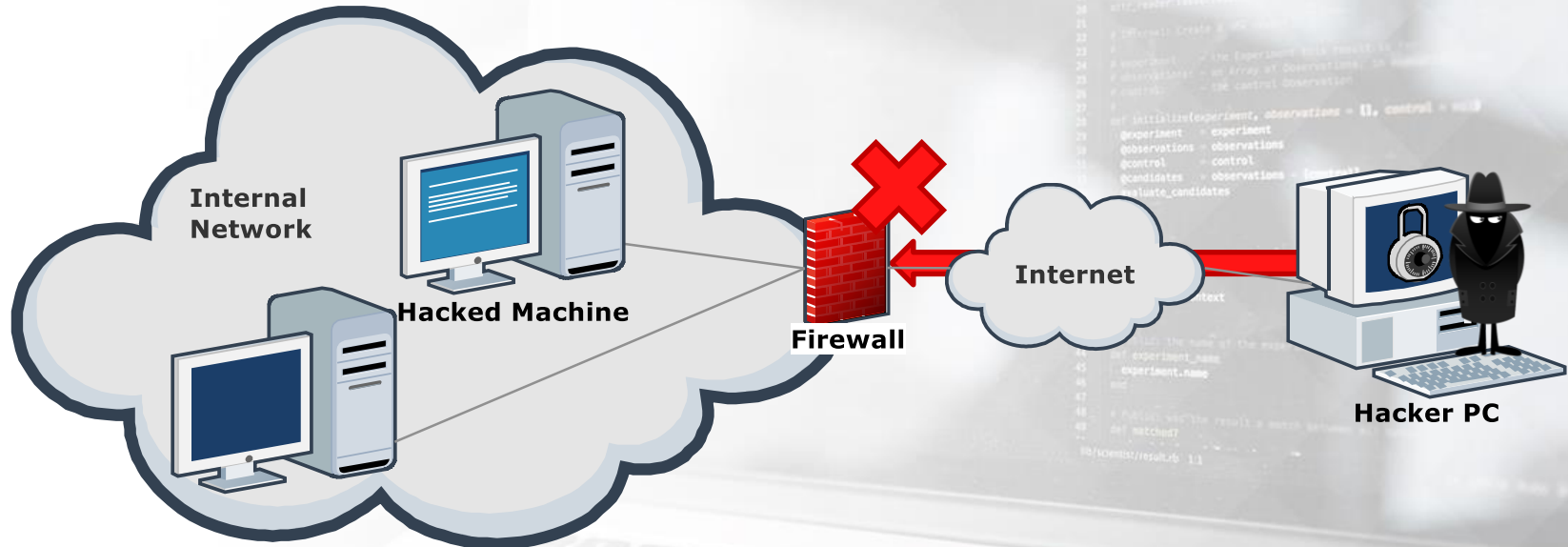
A typical firewall configuration rule is to let HTTP traffic (TCP.dst = 80) pass, so internal workstations can browse the Internet. The remaining traffic is dropped.

What happens to the Trojan horse?



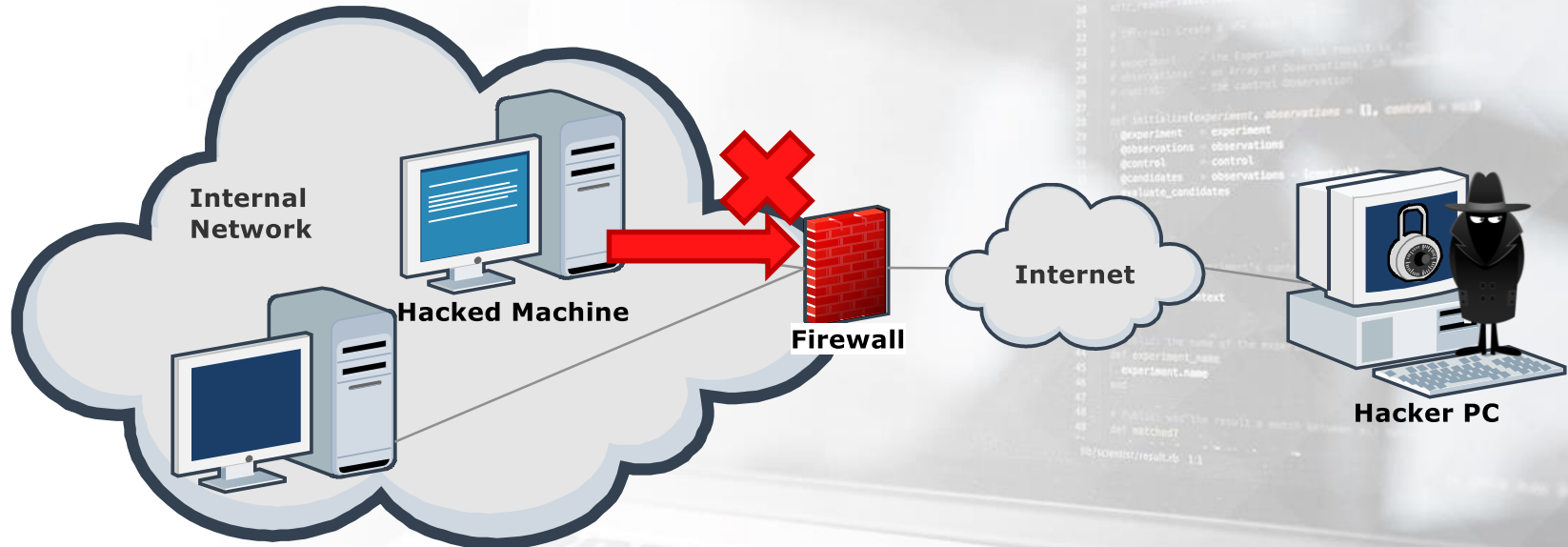
2.6.2.2 Packet Filtering vs. Trojan Horse

The hacker cannot connect to the infected machine, as there are no rules to allow traffic coming to port 123.



2.6.2.2 Packet Filtering vs. Trojan Horse

Similarly, the firewall will deny the traffic from the infected machine to the hacker's PC.



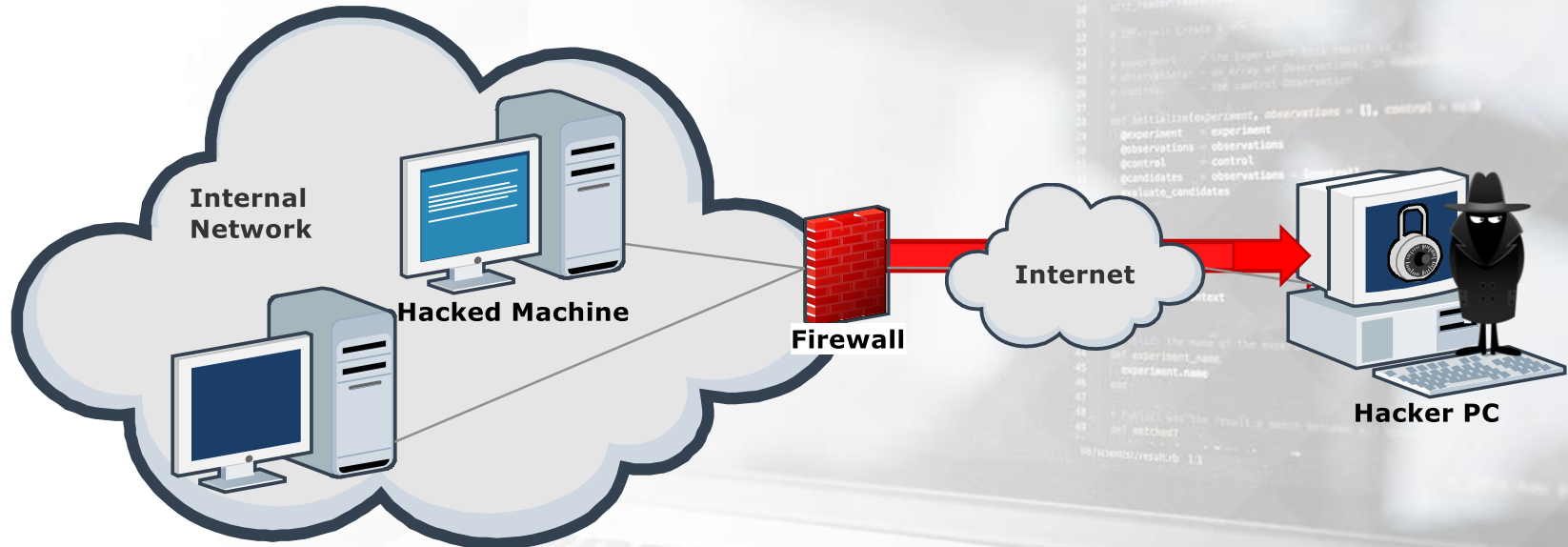
2.6.2.2 Packet Filtering vs. Trojan Horse

What happens if the hacker, who is smart and knows all the well-known ports by heart, configures the Trojan to **connect back** to his or her machine on port 80?



2.6.2.2 Packet Filtering vs. Trojan Horse

The firewall will **allow** the connection!



2.6.3 Application Layer Firewalls

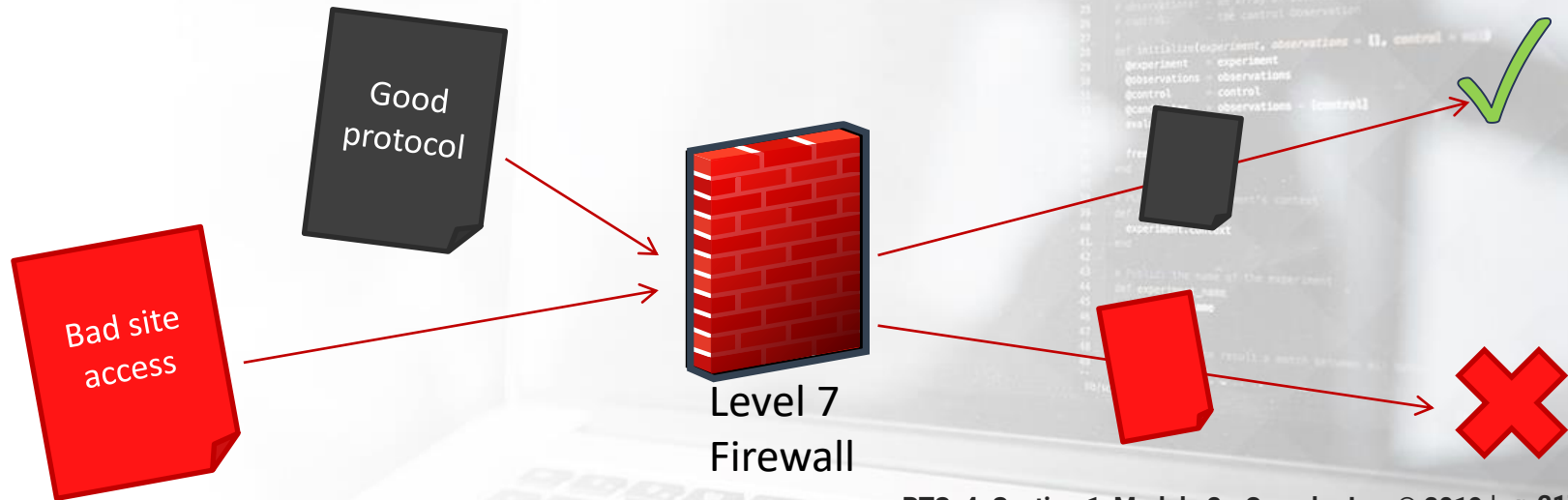
Application level firewalls work by checking all the OSI 7 layers.

They provide a more comprehensive protection because they inspect the actual content of a packet, not just its headers. For example:

- Drop any peer-to-peer application packet.
- Prevent users from visiting a site.

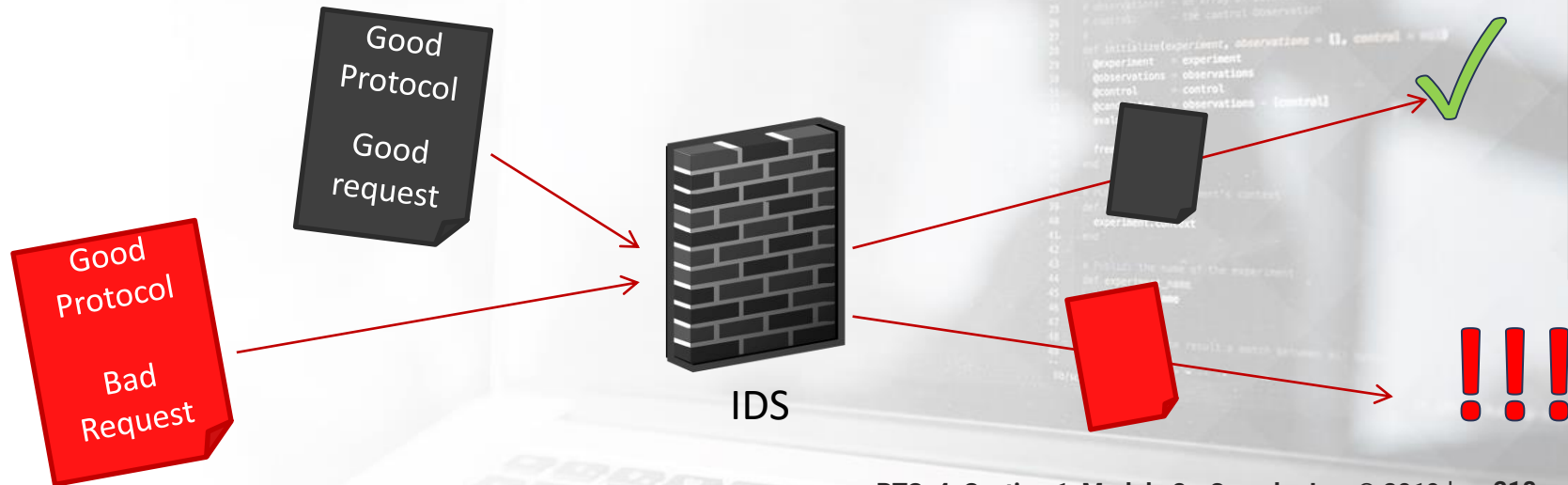
2.6.3 Application Layer Firewalls

Level 7 firewalls are indeed able to understand most of the application layer protocols in use nowadays. Organizations use them not only to protect their network from hackers but also to filter unwanted traffic.



2.6.4 IDS

There is not just traffic detection, but intrusion detection!
Intrusion Detection Systems (IDS) inspect the application payload trying to detect any potential attack.



2.6.4 IDS

An IDS is specialized software used for **detecting ongoing intrusions**. It checks for attack vectors like ping sweeps, port scans, SQL injections, buffer overflows and so on.

IDS can also **identify traffic** generated by a virus or a worm. Pretty much every kind of network threat can be detected by a well-configured IDS.



2.6.4 IDS

An IDS, like an antivirus, detects risky traffic by means of **signatures**. The vendor provides frequent signature updates as soon as new attack vectors are found in the wild. Without the right signatures an IDS cannot detect and report an intrusion; the **IDS cannot detect something if it does not already know**.

There are also **false positives**. They happen when legit traffic is flagged as malicious.



2.6.4 IDS

Detection is performed by a multitude of **sensors**, software components that inspect network traffic.

Sensors passively intercept intrusions and communicate them to the **IDS manager**, software in charge of maintaining policies and which provides a management console to the system administrator.

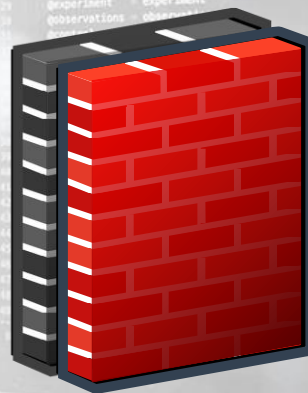
```
14 def __init__(self, name, ip, port):
15     self.name = name
16     self.ip = ip
17     self.port = port
18
19     # The experiment will result in a
20     # list of observations, an array of
21     # observations
22     self.observations = []
23     self.control = None
24
25     def initialize(self, experiment, observations = [], control = None):
26         self.experiment = experiment
27         self.observations = observations
28         self.control = control
29         self.evaluate_candidates()
30
31     def evaluate_candidates(self):
32         # Evaluate the experiment's control
33         self.control = self.experiment.evaluate_candidates(
34             self.observations)
35
36     def get_name(self):
37         return self.name
38
39     def get_ip(self):
40         return self.ip
41
42     def get_port(self):
43         return self.port
44
45     def get_name(self):
46         return self.name
47
48     def get_ip(self):
49         return self.ip
50
51     def get_port(self):
52         return self.port
53
54     def get_name(self):
55         return self.name
56
57     def get_ip(self):
58         return self.ip
59
60     def get_port(self):
61         return self.port
62
63     def get_name(self):
64         return self.name
65
66     def get_ip(self):
67         return self.ip
68
69     def get_port(self):
70         return self.port
71
72     def get_name(self):
73         return self.name
74
75     def get_ip(self):
76         return self.ip
77
78     def get_port(self):
79         return self.port
80
81     def get_name(self):
82         return self.name
83
84     def get_ip(self):
85         return self.ip
86
87     def get_port(self):
88         return self.port
89
90     def get_name(self):
91         return self.name
92
93     def get_ip(self):
94         return self.ip
95
96     def get_port(self):
97         return self.port
98
99     def get_name(self):
100         return self.name
101
102     def get_ip(self):
103         return self.ip
104
105     def get_port(self):
106         return self.port
107
108     def get_name(self):
109         return self.name
110
111     def get_ip(self):
112         return self.ip
113
114     def get_port(self):
115         return self.port
116
117     def get_name(self):
118         return self.name
119
120     def get_ip(self):
121         return self.ip
122
123     def get_port(self):
124         return self.port
125
126     def get_name(self):
127         return self.name
128
129     def get_ip(self):
130         return self.ip
131
132     def get_port(self):
133         return self.port
134
135     def get_name(self):
136         return self.name
137
138     def get_ip(self):
139         return self.ip
140
141     def get_port(self):
142         return self.port
143
144     def get_name(self):
145         return self.name
146
147     def get_ip(self):
148         return self.ip
149
150     def get_port(self):
151         return self.port
152
153     def get_name(self):
154         return self.name
155
156     def get_ip(self):
157         return self.ip
158
159     def get_port(self):
160         return self.port
161
162     def get_name(self):
163         return self.name
164
165     def get_ip(self):
166         return self.ip
167
168     def get_port(self):
169         return self.port
170
171     def get_name(self):
172         return self.name
173
174     def get_ip(self):
175         return self.ip
176
177     def get_port(self):
178         return self.port
179
180     def get_name(self):
181         return self.name
182
183     def get_ip(self):
184         return self.ip
185
186     def get_port(self):
187         return self.port
188
189     def get_name(self):
190         return self.name
191
192     def get_ip(self):
193         return self.ip
194
195     def get_port(self):
196         return self.port
197
198     def get_name(self):
199         return self.name
200
201     def get_ip(self):
202         return self.ip
203
204     def get_port(self):
205         return self.port
206
207     def get_name(self):
208         return self.name
209
210     def get_ip(self):
211         return self.ip
212
213     def get_port(self):
214         return self.port
215
216     def get_name(self):
217         return self.name
218
219     def get_ip(self):
220         return self.ip
221
222     def get_port(self):
223         return self.port
224
225     def get_name(self):
226         return self.name
227
228     def get_ip(self):
229         return self.ip
230
231     def get_port(self):
232         return self.port
233
234     def get_name(self):
235         return self.name
236
237     def get_ip(self):
238         return self.ip
239
240     def get_port(self):
241         return self.port
242
243     def get_name(self):
244         return self.name
245
246     def get_ip(self):
247         return self.ip
248
249     def get_port(self):
250         return self.port
251
252     def get_name(self):
253         return self.name
254
255     def get_ip(self):
256         return self.ip
257
258     def get_port(self):
259         return self.port
260
261     def get_name(self):
262         return self.name
263
264     def get_ip(self):
265         return self.ip
266
267     def get_port(self):
268         return self.port
269
270     def get_name(self):
271         return self.name
272
273     def get_ip(self):
274         return self.ip
275
276     def get_port(self):
277         return self.port
278
279     def get_name(self):
280         return self.name
281
282     def get_ip(self):
283         return self.ip
284
285     def get_port(self):
286         return self.port
287
288     def get_name(self):
289         return self.name
290
291     def get_ip(self):
292         return self.ip
293
294     def get_port(self):
295         return self.port
296
297     def get_name(self):
298         return self.name
299
300     def get_ip(self):
301         return self.ip
302
303     def get_port(self):
304         return self.port
305
306     def get_name(self):
307         return self.name
308
309     def get_ip(self):
310         return self.ip
311
312     def get_port(self):
313         return self.port
314
315     def get_name(self):
316         return self.name
317
318     def get_ip(self):
319         return self.ip
320
321     def get_port(self):
322         return self.port
323
324     def get_name(self):
325         return self.name
326
327     def get_ip(self):
328         return self.ip
329
330     def get_port(self):
331         return self.port
332
333     def get_name(self):
334         return self.name
335
336     def get_ip(self):
337         return self.ip
338
339     def get_port(self):
340         return self.port
341
342     def get_name(self):
343         return self.name
344
345     def get_ip(self):
346         return self.ip
347
348     def get_port(self):
349         return self.port
350
351     def get_name(self):
352         return self.name
353
354     def get_ip(self):
355         return self.ip
356
357     def get_port(self):
358         return self.port
359
360     def get_name(self):
361         return self.name
362
363     def get_ip(self):
364         return self.ip
365
366     def get_port(self):
367         return self.port
368
369     def get_name(self):
370         return self.name
371
372     def get_ip(self):
373         return self.ip
374
375     def get_port(self):
376         return self.port
377
378     def get_name(self):
379         return self.name
380
381     def get_ip(self):
382         return self.ip
383
384     def get_port(self):
385         return self.port
386
387     def get_name(self):
388         return self.name
389
390     def get_ip(self):
391         return self.ip
392
393     def get_port(self):
394         return self.port
395
396     def get_name(self):
397         return self.name
398
399     def get_ip(self):
400         return self.ip
401
402     def get_port(self):
403         return self.port
404
405     def get_name(self):
406         return self.name
407
408     def get_ip(self):
409         return self.ip
410
411     def get_port(self):
412         return self.port
413
414     def get_name(self):
415         return self.name
416
417     def get_ip(self):
418         return self.ip
419
420     def get_port(self):
421         return self.port
422
423     def get_name(self):
424         return self.name
425
426     def get_ip(self):
427         return self.ip
428
429     def get_port(self):
430         return self.port
431
432     def get_name(self):
433         return self.name
434
435     def get_ip(self):
436         return self.ip
437
438     def get_port(self):
439         return self.port
440
441     def get_name(self):
442         return self.name
443
444     def get_ip(self):
445         return self.ip
446
447     def get_port(self):
448         return self.port
449
450     def get_name(self):
451         return self.name
452
453     def get_ip(self):
454         return self.ip
455
456     def get_port(self):
457         return self.port
458
459     def get_name(self):
460         return self.name
461
462     def get_ip(self):
463         return self.ip
464
465     def get_port(self):
466         return self.port
467
468     def get_name(self):
469         return self.name
470
471     def get_ip(self):
472         return self.ip
473
474     def get_port(self):
475         return self.port
476
477     def get_name(self):
478         return self.name
479
480     def get_ip(self):
481         return self.ip
482
483     def get_port(self):
484         return self.port
485
486     def get_name(self):
487         return self.name
488
489     def get_ip(self):
490         return self.ip
491
492     def get_port(self):
493         return self.port
494
495     def get_name(self):
496         return self.name
497
498     def get_ip(self):
499         return self.ip
500
501     def get_port(self):
502         return self.port
503
504     def get_name(self):
505         return self.name
506
507     def get_ip(self):
508         return self.ip
509
510     def get_port(self):
511         return self.port
512
513     def get_name(self):
514         return self.name
515
516     def get_ip(self):
517         return self.ip
518
519     def get_port(self):
520         return self.port
521
522     def get_name(self):
523         return self.name
524
525     def get_ip(self):
526         return self.ip
527
528     def get_port(self):
529         return self.port
530
531     def get_name(self):
532         return self.name
533
534     def get_ip(self):
535         return self.ip
536
537     def get_port(self):
538         return self.port
539
540     def get_name(self):
541         return self.name
542
543     def get_ip(self):
544         return self.ip
545
546     def get_port(self):
547         return self.port
548
549     def get_name(self):
550         return self.name
551
552     def get_ip(self):
553         return self.ip
554
555     def get_port(self):
556         return self.port
557
558     def get_name(self):
559         return self.name
560
561     def get_ip(self):
562         return self.ip
563
564     def get_port(self):
565         return self.port
566
567     def get_name(self):
568         return self.name
569
570     def get_ip(self):
571         return self.ip
572
573     def get_port(self):
574         return self.port
575
576     def get_name(self):
577         return self.name
578
579     def get_ip(self):
580         return self.ip
581
582     def get_port(self):
583         return self.port
584
585     def get_name(self):
586         return self.name
587
588     def get_ip(self):
589         return self.ip
590
591     def get_port(self):
592         return self.port
593
594     def get_name(self):
595         return self.name
596
597     def get_ip(self):
598         return self.ip
599
600     def get_port(self):
601         return self.port
602
603     def get_name(self):
604         return self.name
605
606     def get_ip(self):
607         return self.ip
608
609     def get_port(self):
610         return self.port
611
612     def get_name(self):
613         return self.name
614
615     def get_ip(self):
616         return self.ip
617
618     def get_port(self):
619         return self.port
620
621     def get_name(self):
622         return self.name
623
624     def get_ip(self):
625         return self.ip
626
627     def get_port(self):
628         return self.port
629
630     def get_name(self):
631         return self.name
632
633     def get_ip(self):
634         return self.ip
635
636     def get_port(self):
637         return self.port
638
639     def get_name(self):
640         return self.name
641
642     def get_ip(self):
643         return self.ip
644
645     def get_port(self):
646         return self.port
647
648     def get_name(self):
649         return self.name
650
651     def get_ip(self):
652         return self.ip
653
654     def get_port(self):
655         return self.port
656
657     def get_name(self):
658         return self.name
659
660     def get_ip(self):
661         return self.ip
662
663     def get_port(self):
664         return self.port
665
666     def get_name(self):
667         return self.name
668
669     def get_ip(self):
670         return self.ip
671
672     def get_port(self):
673         return self.port
674
675     def get_name(self):
676         return self.name
677
678     def get_ip(self):
679         return self.ip
680
681     def get_port(self):
682         return self.port
683
684     def get_name(self):
685         return self.name
686
687     def get_ip(self):
688         return self.ip
689
690     def get_port(self):
691         return self.port
692
693     def get_name(self):
694         return self.name
695
696     def get_ip(self):
697         return self.ip
698
699     def get_port(self):
700         return self.port
701
702     def get_name(self):
703         return self.name
704
705     def get_ip(self):
706         return self.ip
707
708     def get_port(self):
709         return self.port
710
711     def get_name(self):
712         return self.name
713
714     def get_ip(self):
715         return self.ip
716
717     def get_port(self):
718         return self.port
719
720     def get_name(self):
721         return self.name
722
723     def get_ip(self):
724         return self.ip
725
726     def get_port(self):
727         return self.port
728
729     def get_name(self):
730         return self.name
731
732     def get_ip(self):
733         return self.ip
734
735     def get_port(self):
736         return self.port
737
738     def get_name(self):
739         return self.name
740
741     def get_ip(self):
742         return self.ip
743
744     def get_port(self):
745         return self.port
746
747     def get_name(self):
748         return self.name
749
750     def get_ip(self):
751         return self.ip
752
753     def get_port(self):
754         return self.port
755
756     def get_name(self):
757         return self.name
758
759     def get_ip(self):
760         return self.ip
761
762     def get_port(self):
763         return self.port
764
765     def get_name(self):
766         return self.name
767
768     def get_ip(self):
769         return self.ip
770
771     def get_port(self):
772         return self.port
773
774     def get_name(self):
775         return self.name
776
777     def get_ip(self):
778         return self.ip
779
780     def get_port(self):
781         return self.port
782
783     def get_name(self):
784         return self.name
785
786     def get_ip(self):
787         return self.ip
788
789     def get_port(self):
790         return self.port
791
792     def get_name(self):
793         return self.name
794
795     def get_ip(self):
796         return self.ip
797
798     def get_port(self):
799         return self.port
800
801     def get_name(self):
802         return self.name
803
804     def get_ip(self):
805         return self.ip
806
807     def get_port(self):
808         return self.port
809
810     def get_name(self):
811         return self.name
812
813     def get_ip(self):
814         return self.ip
815
816     def get_port(self):
817         return self.port
818
819     def get_name(self):
820         return self.name
821
822     def get_ip(self):
823         return self.ip
824
825     def get_port(self):
826         return self.port
827
828     def get_name(self):
829         return self.name
830
831     def get_ip(self):
832         return self.ip
833
834     def get_port(self):
835         return self.port
836
837     def get_name(self):
838         return self.name
839
840     def get_ip(self):
841         return self.ip
842
843     def get_port(self):
844         return self.port
845
846     def get_name(self):
847         return self.name
848
849     def get_ip(self):
850         return self.ip
851
852     def get_port(self):
853         return self.port
854
855     def get_name(self):
856         return self.name
857
858     def get_ip(self):
859         return self.ip
860
861     def get_port(self):
862         return self.port
863
864     def get_name(self):
865         return self.name
866
867     def get_ip(self):
868         return self.ip
869
870     def get_port(self):
871         return self.port
872
873     def get_name(self):
874         return self.name
875
876     def get_ip(self):
877         return self.ip
878
879     def get_port(self):
880         return self.port
881
882     def get_name(self):
883         return self.name
884
885     def get_ip(self):
886         return self.ip
887
888     def get_port(self):
889         return self.port
890
891     def get_name(self):
892         return self.name
893
894     def get_ip(self):
895         return self.ip
896
897     def get_port(self):
898         return self.port
899
900     def get_name(self):
901         return self.name
902
903     def get_ip(self):
904         return self.ip
905
906     def get_port(self):
907         return self.port
908
909     def get_name(self):
910         return self.name
911
912     def get_ip(self):
913         return self.ip
914
915     def get_port(self):
916         return self.port
917
918     def get_name(self):
919         return self.name
920
921     def get_ip(self):
922         return self.ip
923
924     def get_port(self):
925         return self.port
926
927     def get_name(self):
928         return self.name
929
930     def get_ip(self):
931         return self.ip
932
933     def get_port(self):
934         return self.port
935
936     def get_name(self):
937         return self.name
938
939     def get_ip(self):
940         return self.ip
941
942     def get_port(self):
943         return self.port
944
945     def get_name(self):
946         return self.name
947
948     def get_ip(self):
949         return self.ip
950
951     def get_port(self):
952         return self.port
953
954     def get_name(self):
955         return self.name
956
957     def get_ip(self):
958         return self.ip
959
960     def get_port(self):
961         return self.port
962
963     def get_name(self):
964         return self.name
965
966     def get_ip(self):
967         return self.ip
968
969     def get_port(self):
970         return self.port
971
972     def get_name(self):
973         return self.name
974
975     def get_ip(self):
976         return self.ip
977
978     def get_port(self):
979         return self.port
980
981     def get_name(self):
982         return self.name
983
984     def get_ip(self):
985         return self.ip
986
987     def get_port(self):
988         return self.port
989
990     def get_name(self):
991         return self.name
992
993     def get_ip(self):
994         return self.ip
995
996     def get_port(self):
997         return self.port
998
999     def get_name(self):
1000         return self.name
1001
1002     def get_ip(self):
1003         return self.ip
1004
1005     def get_port(self):
1006         return self.port
1007
1008     def get_name(self):
1009         return self.name
1010
1011     def get_ip(self):
1012         return self.ip
1013
1014     def get_port(self):
1015         return self.port
1016
1017     def get_name(self):
1018         return self.name
1019
1020     def get_ip(self):
1021         return self.ip
1022
1023     def get_port(self):
1024         return self.port
1025
1026     def get_name(self):
1027         return self.name
1028
1029     def get_ip(self):
1030         return self.ip
1031
1032     def get_port(self):
1033         return self.port
1034
1035     def get_name(self):
1036         return self.name
1037
1038     def get_ip(self):
1039         return self.ip
1040
1041     def get_port(self):
1042         return self.port
1043
1044     def get_name(self):
1045         return self.name
1046
1047     def get_ip(self):
1048         return self.ip
1049
1050     def get_port(self):
1051         return self.port
1052
1053     def get_name(self):
1054         return self.name
1055
1056     def get_ip(self):
1057         return self.ip
1058
1059     def get_port(self):
1060         return self.port
1061
1062     def get_name(self):
1063         return self.name
1064
1065     def get_ip(self):
1066         return self.ip
1067
1068     def get_port(self):
1069         return self.port
1070
1071     def get_name(self):
1072         return self.name
1073
1074     def get_ip(self):
1075         return self.ip
1076
1077     def get_port(self):
1078         return self.port
1079
1080     def get_name(self):
1081         return self.name
1082
1083     def get_ip(self):
1084         return self.ip
1085
1086     def get_port(self):
1087         return self.port
1088
1089     def get_name(self):
1090         return self.name
1091
1092     def get_ip(self):
1093         return self.ip
1094
1095     def get_port(self):
1096         return self.port
1097
1098     def get_name(self):
1099         return self.name
1100
1101     def get_ip(self):
1102         return self.ip
1103
1104     def get_port(self):
1105         return self.port
1106
1107     def get_name(self):
1108         return self.name
1109
1110     def get_ip(self):
1111         return self.ip
1112
1113     def get_port(self):
1114         return self.port
1115
1116     def get_name(self):
1117         return self.name
1118
1119     def get_ip(self):
1120         return self.ip
1121
1122     def get_port(self):
1123         return self.port
1124
1125     def get_name(self):
1126         return self.name
1127
1128     def get_ip(self):
1129         return self.ip
1130
1131     def get_port(self):
1132         return self.port
1133
1134     def get_name(self):
1135         return self.name
1136
1137     def get_ip(self):
1138         return self.ip
1139
1140     def get_port(self):
1141         return self.port
1142
1143     def get_name(self):
1144         return self.name
1145
1146     def get_ip(self):
1147         return self.ip
1148
1149     def get_port(self):
1150         return self.port
1151
1152     def get_name(self):
1153         return self.name
1154
1155     def get_ip(self):
1156         return self.ip
1157
1158     def get_port(self):
1159         return self.port
1160
1161     def get_name(self):
1162         return self.name
1163
1164     def get_ip(self):
1165         return self.ip
1166
1167     def get_port(self):
1168         return self.port
1169
1170     def get_name(self):
1171         return self.name
1172
1173     def get_ip(self):
1174         return self.ip
1175
1176     def get_port(self):
1177         return self.port
1178
1179     def get_name(self):
1180         return self.name
1181
1182     def get_ip(self):
1183         return self.ip
1184
1185     def get_port(self):
1186         return self.port
1187
1188     def get_name(self):
1189         return self.name
1190
1191     def get_ip(self):
1192         return self.ip
1193
1194     def get_port(self):
1195         return self.port
1196
1197     def get_name(self):
1198         return self.name
1199
1200     def get_ip(self):
1201         return self.ip
1202
1203     def get_port(self):
1204         return self.port
1205
1206     def get_name(self):
1207         return self.name
1208
1209     def get_ip(self):
1210         return self.ip
1211
1212     def get_port(self):
1213         return self.port
1214
1215     def get_name(self):
1216         return self.name
1217
1218     def get_ip(self):
1219         return self.ip
1220
1221     def get_port(self):
1222         return self.port
1223
1224     def get_name(self):
1225         return self.name
1226
1227     def get_ip(self):
1228         return self.ip
1229
1230     def get_port(self):
1231         return self.port
1232
1233     def get_name(self):
1234         return self.name
1235
1236     def get_ip(self):
1237         return self.ip
1238
1239     def get_port(self):
1240         return self.port
1241
1242     def get_name(self):
1243         return self.name
1244
1245     def get_ip(self):
1246         return self.ip
1247
1248     def get_port(self):
1249         return self.port
1250
1251     def get_name(self):
1252         return self.name
1253
1254     def get_ip(self):
1255         return self.ip
1256
1257     def get_port(self):
1258         return self.port
1259
1260     def get_name(self):
1261         return self.name
1262
1263     def get_ip(self):
1264         return self.ip
1265
1266     def get_port(self):
1267         return self.port
1268
1269     def get_name(self):
1270         return self.name
1271
1272     def get_ip(self):
1273         return self.ip
1274
1275     def get_port(self):
1276         return self.port
1277
1278     def get_name(self):
1279         return self.name
1280
1281     def get_ip(self):
1282         return self.ip
1283
1284     def get_port(self):
1285         return self.port
1286
1287     def get_name(self):
1288         return self.name
1289
1290     def get_ip(self):
1291         return self.ip
1292
1293     def get_port(self):
1294         return self.port
1295
1296     def get_name(self):
1297         return self.name
1298
1299     def get_ip(self):
1300         return self.ip
1301
1302     def get_port(self):
1303         return self.port
1304
1305     def get_name(self):
1306         return self.name
1307
1308     def get_ip(self):
1309         return self.ip
1310
1311     def get_port(self):
1312         return self.port
1313
1314     def get_name(self):
1315         return self.name
1316
1317     def get_ip(self):
1318         return self.ip
1319
1320     def get_port(self):
1321         return self.port
1322
1323     def get_name(self):
1324         return self.name
1325
1326     def get_ip(self):
1327         return self.ip
1328
1329     def get_port(self):
1330         return self.port
1331
1332     def get_name(self):
1333         return self.name
1334
1335     def get_ip(self):
1336         return self.ip
1337
1338     def get_port(self):
1339         return self.port
1340
1341     def get_name(self):
1342         return self.name
1343
1344     def get_ip(self):
1345         return self.ip
1346
1347     def get_port(self):
1348         return self.port
1349
1350     def get_name(self):
1351         return self.name
1352
1353     def get_ip(self):
1354         return self.ip
1355
1356     def get_port(self):
1357         return self.port
1358
1359     def get_name(self):
1360         return self.name
1361
1362     def get_ip(self):
1363         return self.ip
1364
1365     def get_port(self):
1366         return self.port
1367
1368     def get_name(self):
1369         return self.name
1370
1371     def get_ip(self):
1372         return self.ip
1373
1374     def get_port(self):
1375         return self.port
1376
1377     def get_name(self):
1378         return self.name
1379
1380     def get_ip(self):
1381         return self.ip
1382
1383     def get_port(self):
1384         return self.port
1385
1386     def get_name(self):
1387         return self.name
1388
1389     def get_ip(self):
1390         return self.ip
1391
1392     def get_port(self):
1393         return self.port
1394
1395     def get_name(self):
1396         return self.name
1397
1398     def get_ip(self):
1399         return self.ip
1400
1401     def get_port(self):
1402         return self.port
1403
1404     def get_name(self):
1405         return self.name
1406
1407     def get_ip(self):
1408         return self.ip
1409
1410     def get_port(self):
1411         return self.port
1412
1413     def get_name(self):
1414         return self.name
1415
1416     def get_ip(self):
1417         return self.ip
1418
1419     def get_port(self):
1420         return self.port
1421
1422     def get_name(self):
1423         return self.name
1424
1425     def get_ip(self):
1426         return self.ip
1427
1428     def get_port(self):
1429         return self.port
1430
1431     def get_name(self):
1432         return self.name
1433
1434     def get_ip(self):
1435         return self.ip
1436
1437     def get_port(self):
1438         return self.port
1439
1440     def get_name(self):
1441         return self.name
1442
1443     def get_ip(self):
1444         return self.ip
1445
1446     def get_port(self):
1447         return self.port
1448
1449     def get_name(self):
1450         return self.name
1451
1452     def get_ip(self):
1453         return self.ip
1454
1455     def get_port(self):
1456         return self.port
1457
1458     def get_name(self):
1459         return self.name
1460
1461     def get_ip(self):
1462         return self.ip
1463
1464     def get_port(self):
1465         return self.port
1466
1467     def get_name(self):
1468         return self.name
1469
1470     def get_ip(self):
1471         return self.ip
1472
1473     def get_port(self):
1474         return self.port
1475
1476     def get_name(self):
1477         return self.name
1478
1479     def get_ip(self):
1480         return self.ip
1481
1482     def get_port(self):
1483         return self.port
1484
1485     def get_name(self):
1486         return self.name
1487
1488     def get_ip(self):
1489         return self.ip
1490
1491     def get_port(self):
1492         return self.port
1493
1494     def get_name(self):
1495         return self.name
1496
1497     def get_ip(self):
1498         return self.ip
1499
1500     def get_port(self):
1501         return self.port
1502
1503     def get_name(self):
1504         return self.name
1505
1506     def get_ip(self):
1507         return self.ip
1508
1509     def get_port(self):
1510         return self.port
1511
1512     def get_name(self):
1513         return self.name
1514
1515     def get_ip(self):
1516         return self.ip
1517
1518     def get_port(self):
1519         return self.port
1520
1521     def get_name(self):
1522         return self.name
1523
1524     def get_ip(self):
1525         return self.ip
1526
1527     def get_port(self):
1528         return self.port
1529
1530     def get_name(self):
1531         return self.name
1532
1533     def get_ip(self):
1534         return self.ip
1535
1536     def get_port(self):
1537         return self.port
1538
1539     def get_name(self):
1540         return self.name
1541
1542     def get_ip(self):
1543         return self.ip
1544
1545     def get_port(self):
1546         return self.port
1547
1548     def get_name(self):
1549         return self.name
1550
1551     def get_ip(self):
1552         return self.ip
1553
1554     def get_port(self):
1555         return self.port
1556
1557     def get_name(self):
1558         return self.name
1559
1560     def get_ip(self):
1561         return self.ip
1562
1563     def get_port(self):
1564         return self.port
1565
1566     def get_name(self):
1567         return self.name
1568
1569     def get_ip(self):
1570         return self.ip
1571
1572     def get_port(self):
1573         return self.port
1574
1575     def get_name(self):
1576         return self.name
1577
1578     def get_ip(self):
1579         return self.ip
1580
1581     def get_port(self):
1582         return self.port
1583
1584     def get_name(self):
1585         return self.name
1586
1587     def get_ip(self):
1588         return self.ip
1589
1590     def get_port(self):
1591         return self.port
1592
1593     def get_name(self):
1594         return self.name
1595
1596     def get_ip(self):
1597         return self.ip
1598
1599     def get_port(self):

```

2.6.4 IDS

IDSs do not substitute firewalls.

They support firewalls by providing a further layer of security protecting the network from mainstream and well-known attack vectors.



2.6.4 IDS

IDSs fall into two main categories:

Network Intrusion
Detection
Systems (NIDS)

Host Intrusion
Detection
Systems (HIDS)



2.6.4.1 NIDS

Network intrusion detection systems inspect network traffic by means of sensors which are usually placed on a router or in a network with a high intrusion risk, like a DMZ.



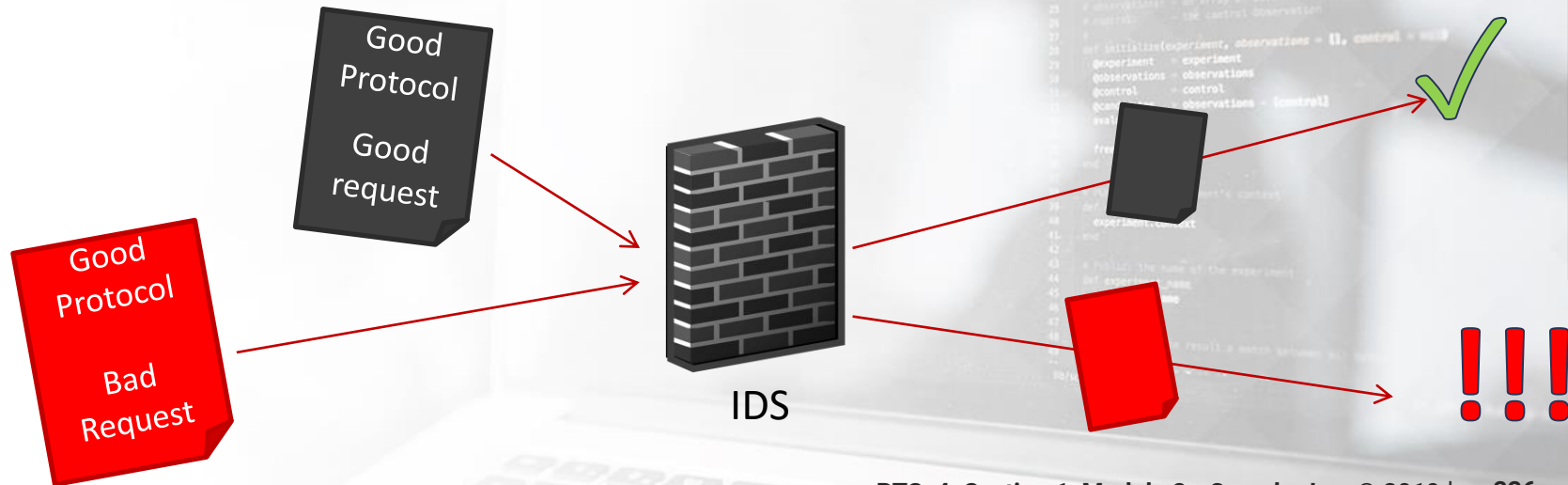
2.6.4.2 HIDS

On the other hand, host IDS sensors monitor application logs, file-system changes and changes to the operating system configuration.



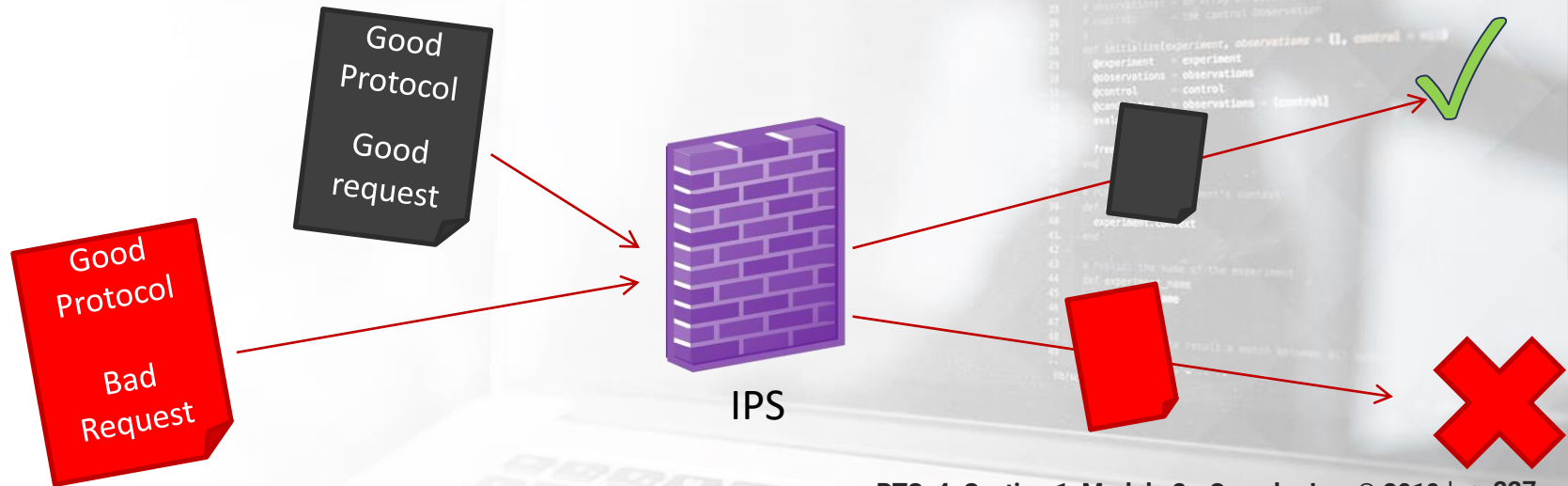
2.6.5 IPS

IDSs, unlike firewalls, can detect suspicious activities and report them to the network administrator. Suspicious activity is logged for future analysis, but **it is not blocked**.



2.6.5 IPS

Intrusion Prevention Systems (IPS) can **drop** malicious requests when the threat has a risk classification above a pre-defined threshold.



2.6.6 Spot an Obstacle

During penetration testing activities, you might want to identify if a firewall-like mechanism is used in the environment.

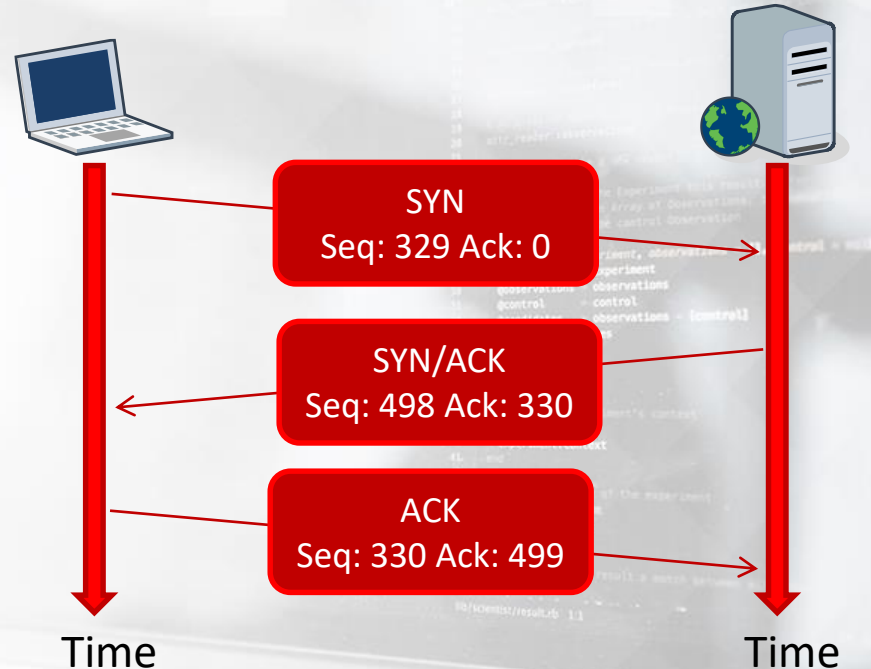
If you suspect presence of a firewall, you might want to check for anomalies in TCP Three-Way Handshake that was introduced previously in this module.

```
21 # Firewall: create a new experiment
22
23 # Note: this is the experiment that will be used
24 # observations - an array of Observations, in this case
25 # control - the control observation
26
27
28 def initialize(experiment, observations = [], control = nil)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations - [control]
33   evaluate_candidates
34
35   # Note: this is the experiment's control
36   per_candidate(experiment)
37
38   # Note: this is the name of the experiment
39
40   def experiment_name
41     @experiment_name
42   end
43
44   # Note: this is the result a match between all
45   def match?
46     # ...
47   end
48
49   # Note: this is the result a match between all
50   def result
51     # ...
52   end
53 end
```



2.6.6 Spot an Obstacle

To recall, proper
Three-way handshake
looks like:

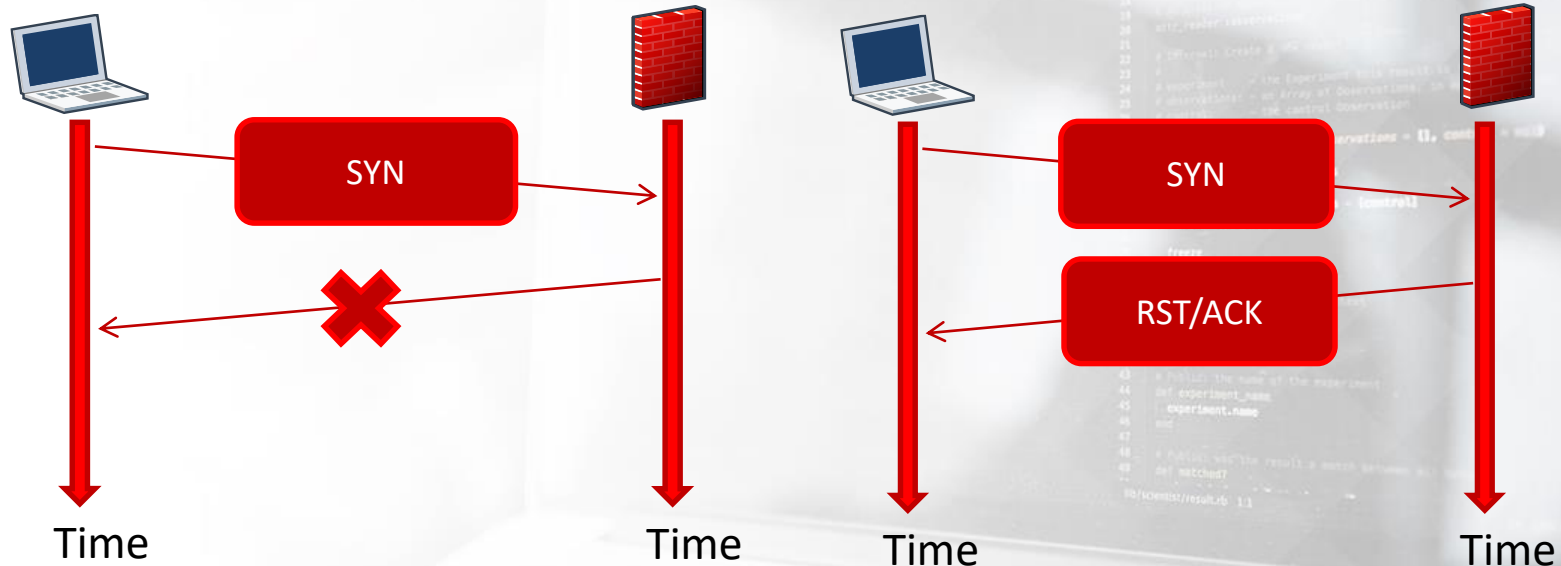


2.6.6 Spot an Obstacle

When a firewall is in place, the following behavior may be spotted:

- **TCP SYN** are sent, but there no **TCP SYN/ACK** replies
- **TCP SYN** packets are sent but a **TCP RST/ACK** reply is received

2.6.6 Spot an Obstacle



2.6.6 Spot an Obstacle

Note, that this type of observation does not determine whether the detected obstacle is a **firewall**, an **IDS**, or **any other** device; this just helps you to identify **environmental constraints**.



2.6.7 NAT and Masquerading

Firewalls not only filter packets but can also be used to implement **Network Address Translation** or **NAT**.

Note: Your home router is most probably running NAT protocol to connect all your home devices to the internet without having to have a public IP assigned for each of them.

What is NAT and why is it needed?

2.6.7 NAT and Masquerading

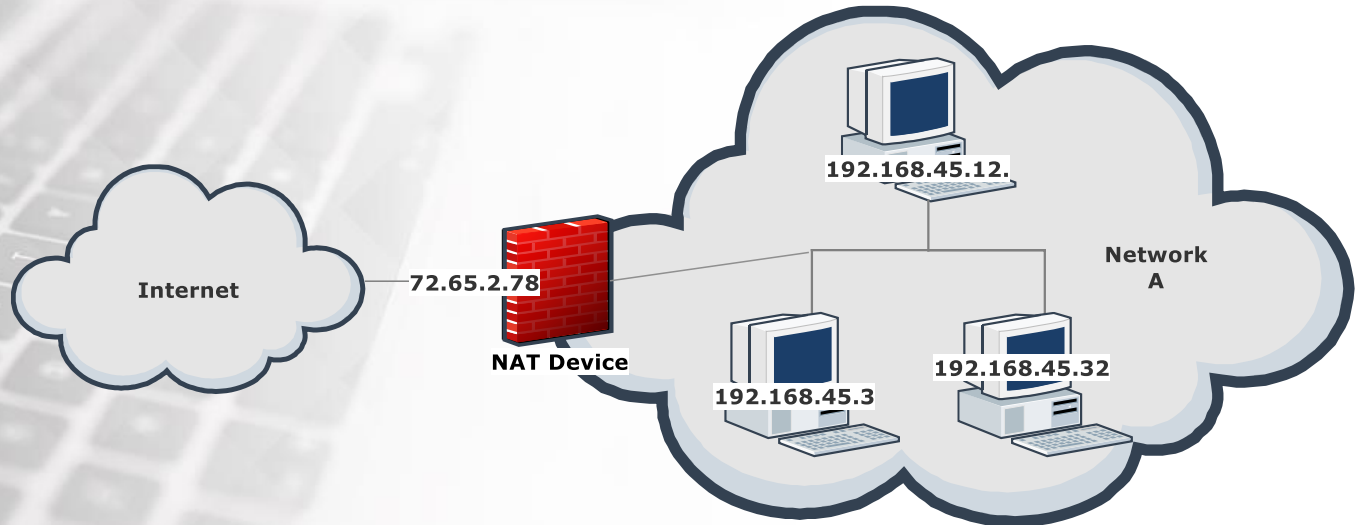
As you know, every machine on the Internet must have a unique IP address. This does not mean that every device that can **access** the internet must have a unique **public** IP address.

Network Address Translation (NAT) and **IP masquerading** are two techniques used to provide access to a network from another network.



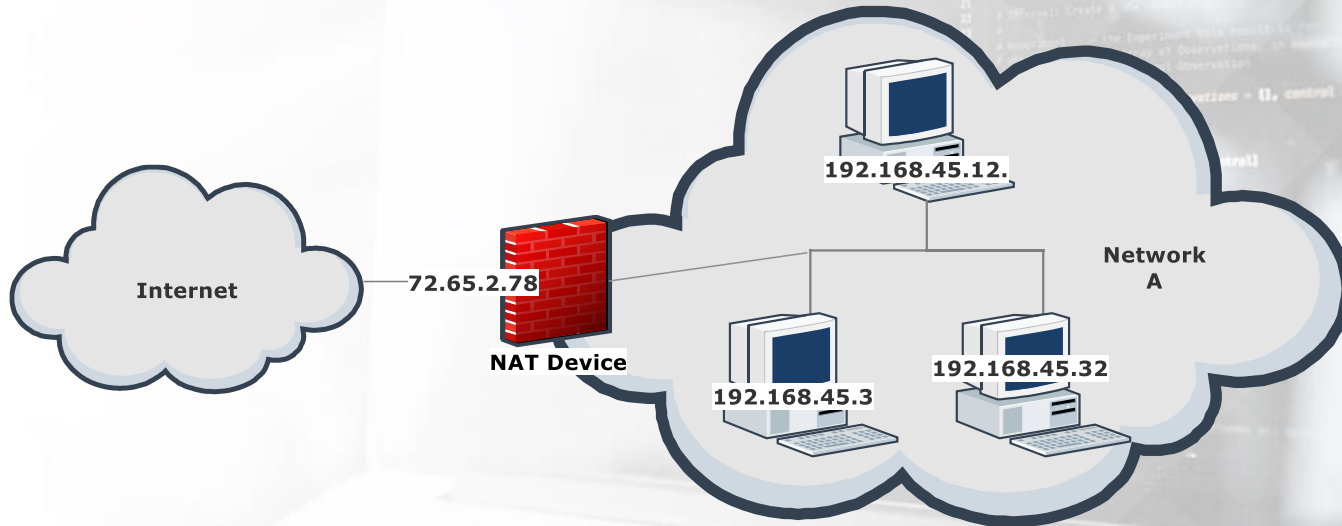
2.6.7 NAT and Masquerading

Network A can be a private network using a NAT device to access the Internet. A machine on the internet cannot directly access a machine in *Network A*.



2.6.7 NAT and Masquerading

But, a machine in *Network A* can access the Internet, if the NAT device allows the traffic to pass.



2.6.7 NAT and Masquerading

Every machine inside *Network A* will use the NAT device as **its default gateway**, thus routing its Internet traffic through it. The NAT device then rewrites the **source IP address** of every packet setting it to `72.65.2.78` (in our example), thus masquerading the original client's IP address.

A machine on the Internet will never know the original client's IP address.



2.6.8 Hera Lab – Find the Secret Server

It is now **practice time!**

In the following lab, you will learn how security through obscurity cannot stop a skilled attacker. Try to solve the challenge by yourself; if you get stuck, you can check the solutions in the member's area.

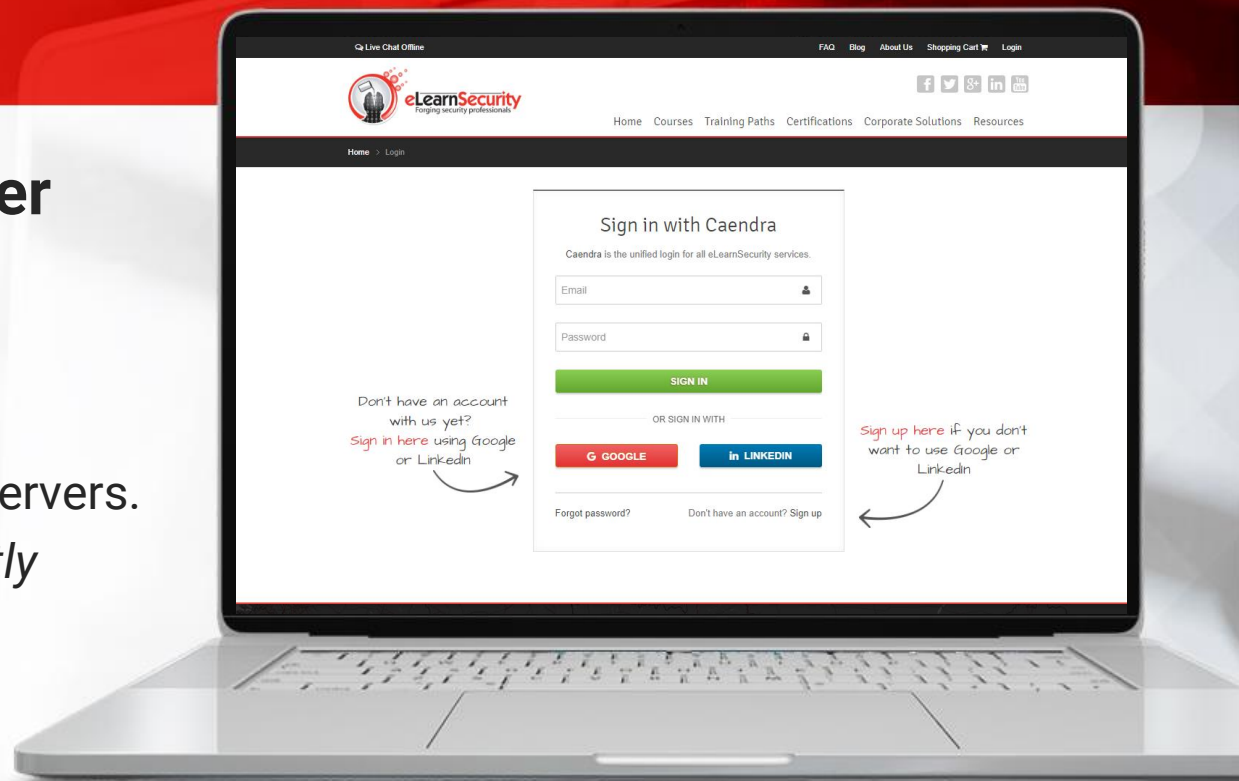


2.6.8 Hera Lab – Find the Secret Server

Find the Secret Server

In this lab, you will:

- Understand routing information.
- Explore different web servers.
- Connect to an *apparently unreachable* server!



**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

2.6.9 Resources

- [Netfilter](#) the Linux kernel packet filtering framework.
- [Book: Firewall Fundamentals](#) the essential guide to understanding and using firewalls to protect personal computers and your network.
- [OSSEC IDS](#) an Open Source Host-based Intrusion Detection System.
- [IDS FAQ](#) answers to simple questions related to detecting intruders who attack systems through the network.

<http://www.netfilter.org/>

http://www.amazon.com/Firewall-Fundamentals-Wes-Noonan/dp/1587052210/ref=cm_cr_pr_sims_t

<http://www.ossec.net/>

http://www.linuxsecurity.com/resource_files/intrusion_detection/network-intrusion-detection.html

DNS



2.7 DNS

How does this support my pentesting career?

- SSL/TLS certificates validation relies on DNS
- Mounting spoofing attacks
- Performing information gathering

2.7 DNS

The **Domain Name System**, or **DNS**, is the only application layer protocol you will see in this module.

The DNS primarily converts human-readable names, like `www.elearnsecurity.com`, to IP addresses and is a fundamental **support protocol** for the Internet and computer networks in general. It is widely recognized that the entire internet security is relying upon DNS.



2.7 DNS

For this course, you will need to know how the DNS service provides name resolution because every common operation on the Internet such as opening a web site, sending an email, and sharing a document involves the use of a DNS to resolve resource names to IP addresses (and vice versa).



2.7.1 DNS Structure

members.elearnsecurity.com

Host

Domain

Top Level
Domain (TLD)

www.sub.domain.com

Host

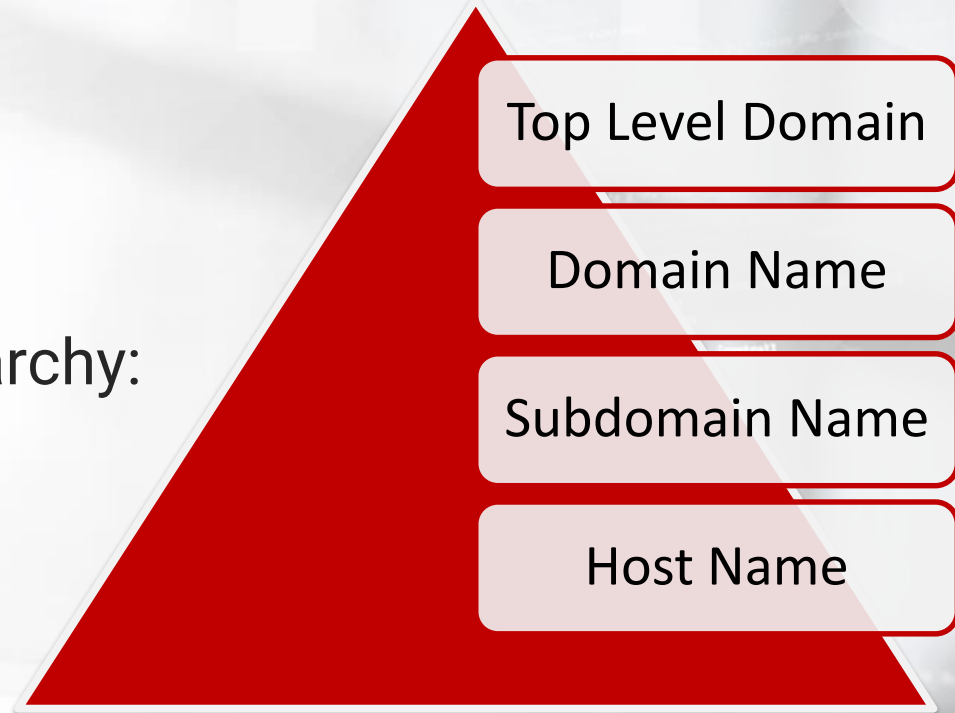
Sub
domain

Domain

Top Level
Domain

2.7.1 DNS Structure

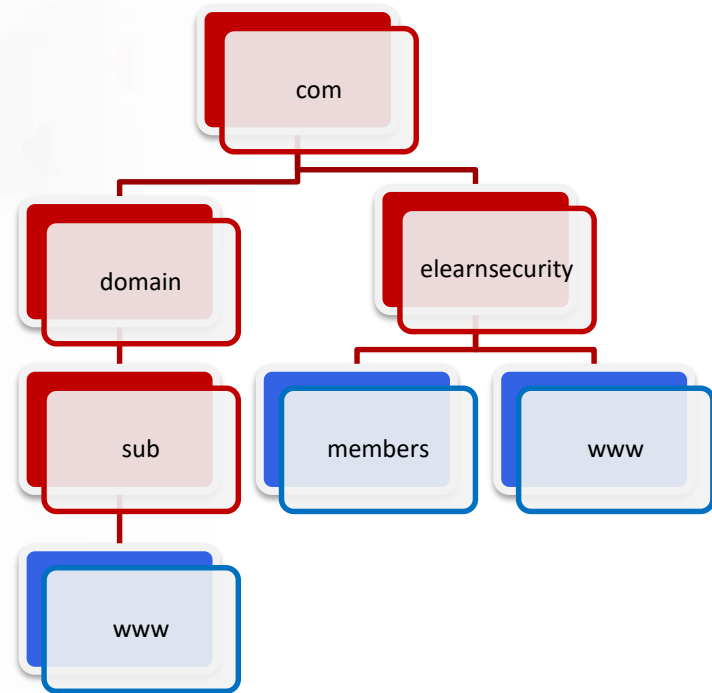
These parts form a hierarchy:



EXAMPLE

2.7.1 DNS Structure

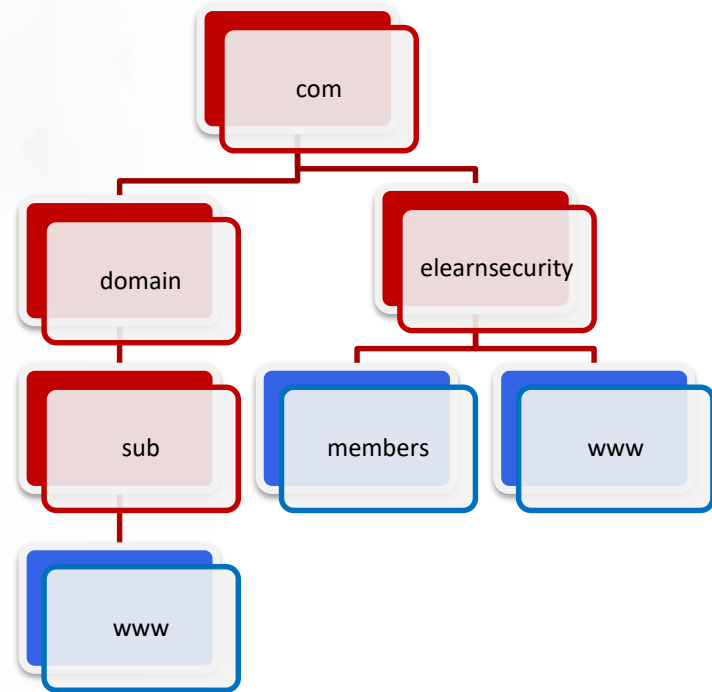
So, we can rewrite the previous names as:



EXAMPLE

2.7.1 DNS Structure

Where the blue squares are the hosts and the red ones are the top/sub/domain names.



2.7.1 DNS Structure

Name resolution is performed by **resolvers**, servers that contact the top level domain (TLD) DNS servers and follow the hierarchy of the DNS name to resolve the name of a host.

Resolvers are DNS servers provided by your ISP or publicly available like OpenDNS or Google DNS.



2.7.2 DNS Names Resolution

To convert a DNS name into an IP address, the operating system must contact a **resolver** server to perform the DNS resolution.

The resolver breaks down the DNS name in its parts and uses them to convert a DNS name into an IP address.

```
24 # Note that the Experiment object is an array of Observations, so we need to
25 # iterate over it to get the control observation
26 # control = the control observation
27
28 def skillsize(experiment, observations = [], control = None)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations - [control]
33   evaluate_candidates
34
35   freeze
36 end
37
38 experiment = Experiment.new
39 experiment.context
40
41 # Create the experiment
42 experiment.name
43
44 # Create the result a match between an experiment and a result
45 def match?
46   # ...
47
48   # ...
49
50   # ...
51   @experiment/result_id
52 end
```

2.7.2.1 DNS Resolution Algorithm

- 1 Firstly, the resolver contacts one of the **root name servers**; these servers contain information about the top level domains.
- 2 Then, it asks the TLD name server what's the name server that can give information (authoritative name server) about the **domain** the resolver is looking for.
- 3 If there are one or more **subdomains**, step 2 is performed again on the authoritative DNS server for every subdomain.
- 4 Finally, the resolver asks for the name resolution of the **host** part.

2.7.2.2 DNS Resolution Example

A computer needs to open a web page on
`www.example.com`.



2.7.2.2 DNS Resolution Example

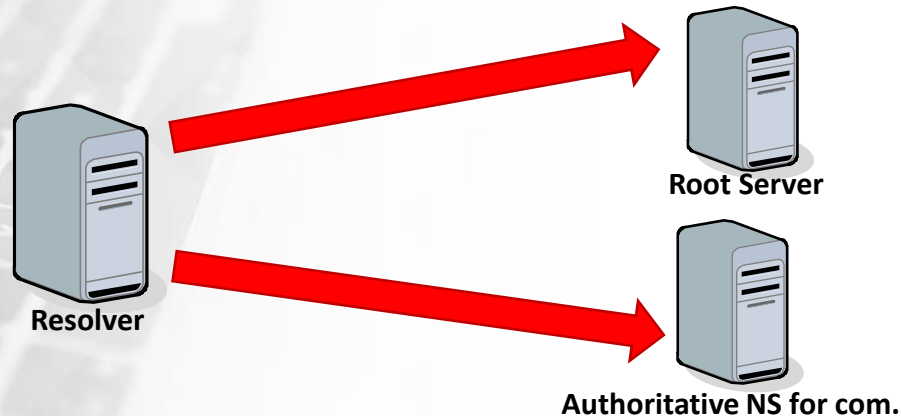
To do that, it contacts the resolver configured by the local administrator (e.g., OpenDNS).



2.7.2.2 DNS Resolution Example

The resolver contacts a **root server** and asks about the authoritative name server(s) for the `com.` domain.

Then the resolver contacts that authoritative name server and asks what is an authoritative name server for the `example.com.` domain.



2.7.2.2 DNS Resolution Example

The resolver then asks what the address of www is.

Finally, the resolver sends the IP address back to the client.



2.7.3 Resolvers and Root Servers

How can a resolver know how to contact a **root name server**?

IP addresses of the root servers are **hardcoded in the configuration** of the resolver. System administrators keep the list updated, otherwise, the resolver would not be able to contact a root server!



2.7.4 Reverse DNS Resolution

The domain name system can also perform the inverse operation; it can convert an **IP address to a DNS name**.

Keep in mind that this is not always the case; the administrator of a domain must have enabled and configured this feature for the domain to make it work.



2.7.4 Reverse DNS Resolution

Many tools use the reverse DNS if it's available.

The Linux ping utility performs a reverse DNS query after receiving every response from the target.



```
$ ping www.yahoo.com
PING fd-fp3.wg1.b.yahoo.com (46.228.47.115) 56(84) bytes of data.
64 bytes from ir1.fp.vip.ir2.yahoo.com (46.228.47.115): icmp_req=1 ttl=49 time=125 ms

--- fd-fp3.wg1.b.yahoo.com ping statistics ---
2 packets transmitted, 1 received, 50% packet loss, time 1001ms
rtt min/avg/max/mdev = 125.706/125.706/125.706/0.000 ms
```



2.7.5 More about the DNS

In this section, you have seen how the DNS is used to perform name resolution. The domain name system is not just that, it is used to identify what the mail servers for a domain are, to know what is the right server for a specific role and much more.

Please refer to [RFC1034](https://www.ietf.org/rfc/rfc1034.txt) and [RFC1035](https://www.ietf.org/rfc/rfc1035.txt) to dig more into DNS!

NOTE: the DNS is also very important to the security of the whole internet because breaking DNS security means breaking SSL and TLS. This, however, is beyond the scope of this training course.

Wireshark



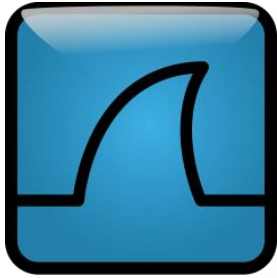
2.8 Wireshark

The best way to deeply understand the topics of this module is to see the actual protocols in action. You can do that by using a sniffer tool.

This section will enhance your Networking and **Wireshark** skills.



2.8 Wireshark



As you know, Wireshark is a network sniffer and protocol analyzer.

This means that you can use it to analyze every packet, traffic stream, or connection **that hits your computer network interface(s)**.

2.8 Wireshark

Knowing this tool is extremely important to understand how networking works.

Wireshark is widely used by network administrators, networking protocol researchers, and hackers.



2.8 Wireshark

Wireshark can capture all the traffic **seen** by the network card of the computer running it.

To understand what traffic a network card sees, you have to know that most network cards, also known as Network Interface Cards (NIC), can work in **promiscuous or monitor mode**.

2.8.1 NIC Promiscuous Mode

During normal operations, a network card **discards** any packet addressed to another NIC. In **promiscuous mode**, a network card will accept and process **any** packet it receives.

For example, in a hub-based network, a NIC will receive traffic addressed to other machines. The NIC usually drops these packets but accepts them while in promiscuous mode.

2.8.1 NIC Promiscuous Mode

With the introduction of switched networks, sniffing other machines Ethernet traffic got harder. You have to perform an attack such as ARP poisoning or MAC flooding in order to do that.

WiFi medium (the air), instead, is broadcast by nature, so it's possible to still detect traffic destined to a different host. In this chapter, we will concentrate on Ethernet traffic only.

```
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

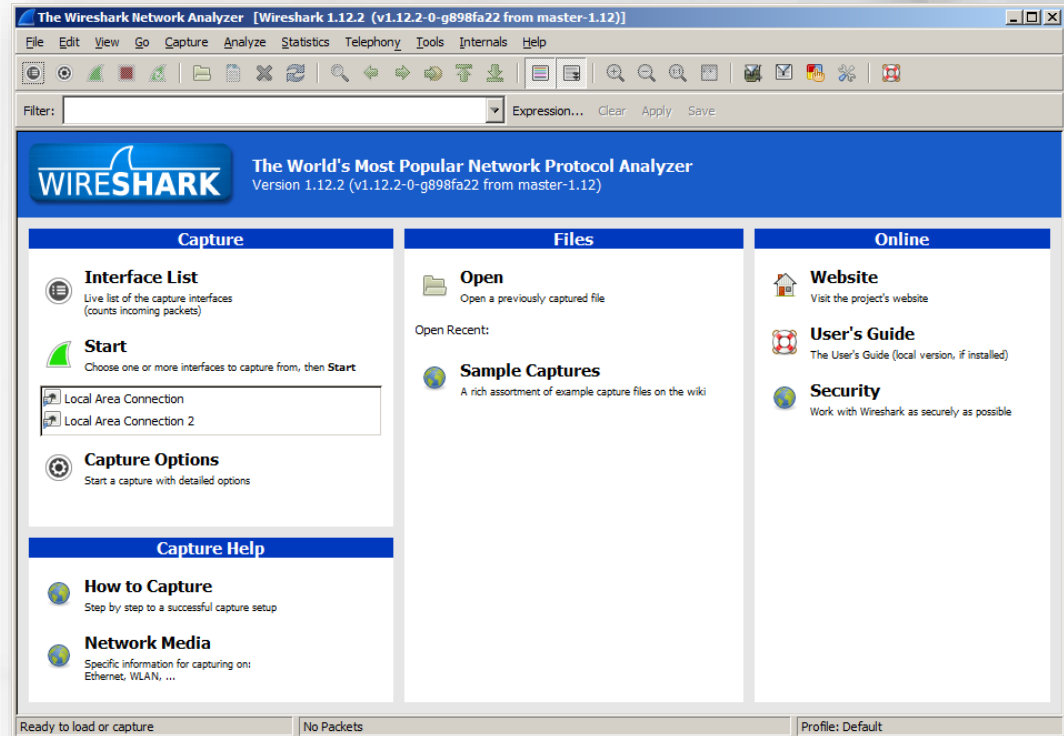
2.8.2 Configuring Wireshark

Wireshark is free software that can run on practically all modern operating systems. You can download it from <https://www.wireshark.org/> or <https://www.wireshark.org/download/> for older versions.

In the following slides, we are going to see how to configure it to use its main features. The version covered is 1.12.2.

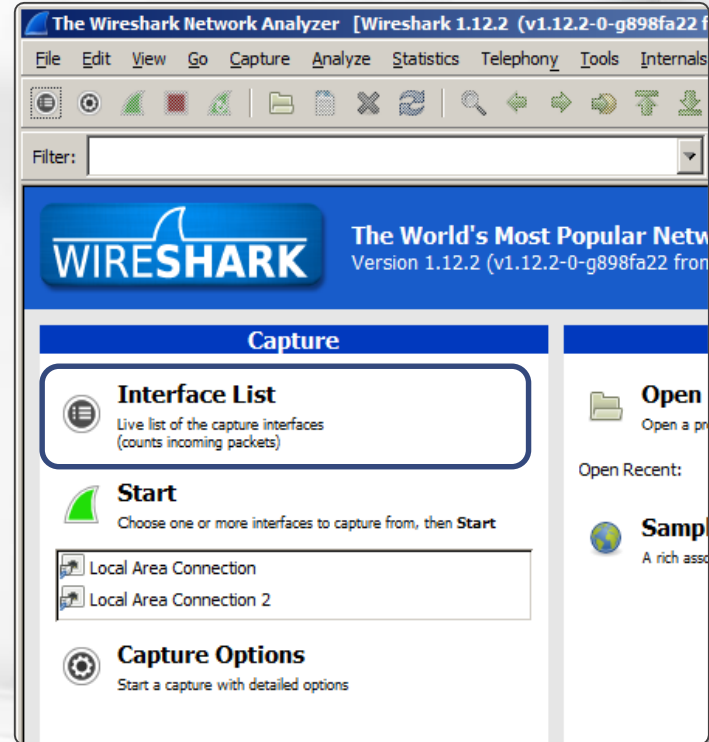
2.8.2 Configuring Wireshark

Here we see
Wireshark's main
window.



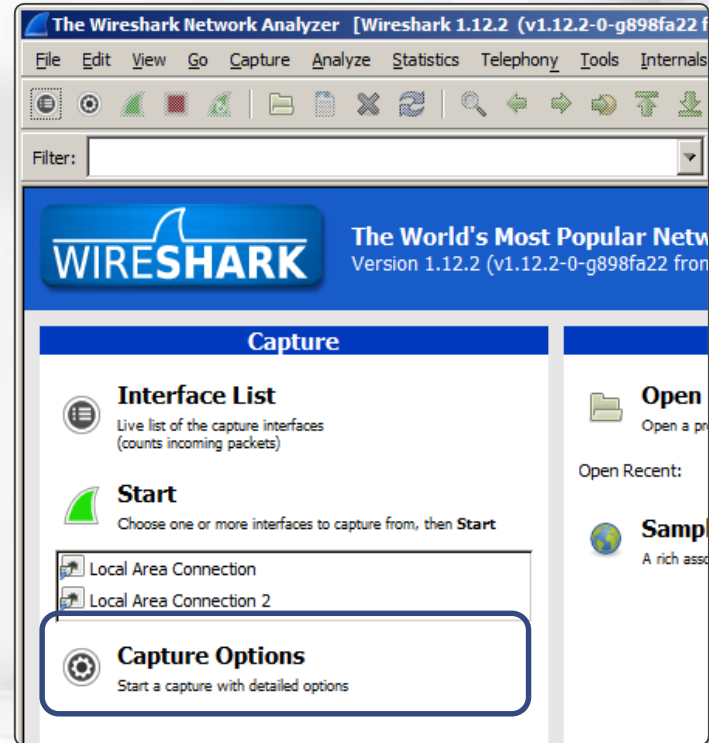
2.8.2 Configuring Wireshark

Clicking on **Interface List** opens a window with a list of your network cards (wired, wireless, VPNs, virtual interfaces, etc.).



2.8.2 Configuring Wireshark

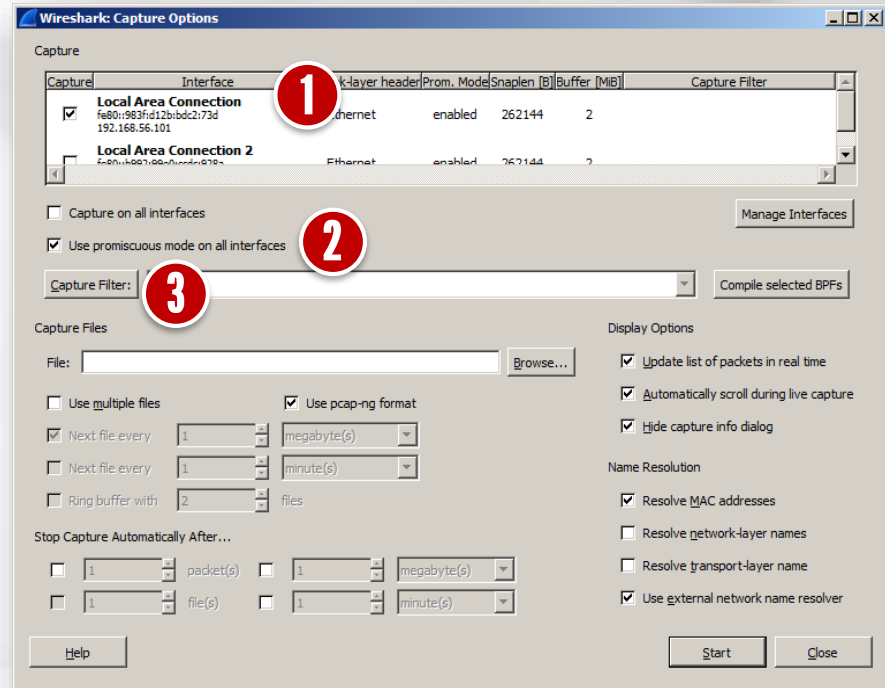
Clicking on **Capture Options** opens...



2.8.2 Configuring Wireshark

... the **capture options** window, which has a huge impact on your capture session as you can configure:

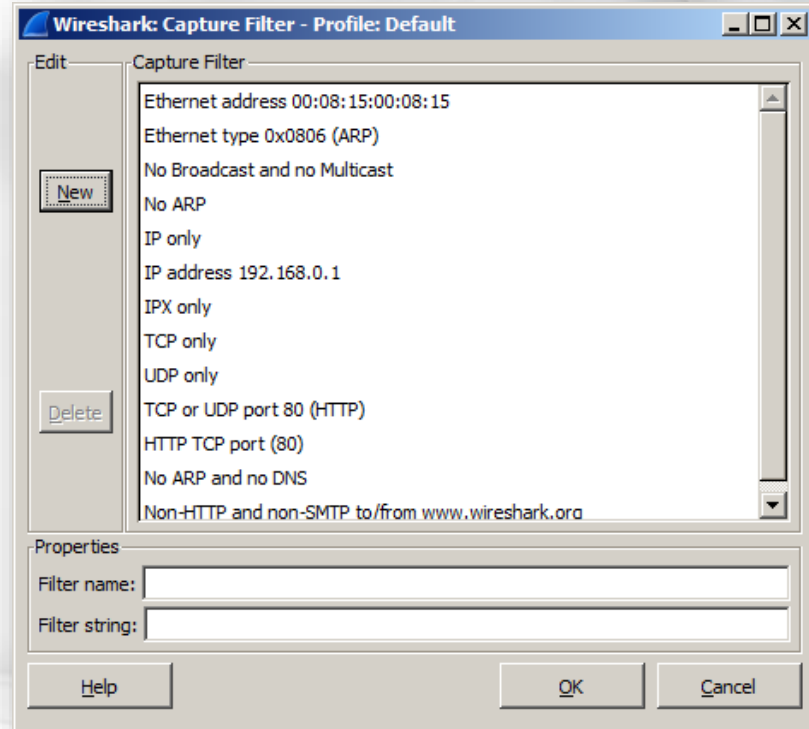
- 1 Which interfaces to use during the capture
- 2 NIC promiscuous mode
- 3 Capture filtering



2.8.2 Configuring Wireshark

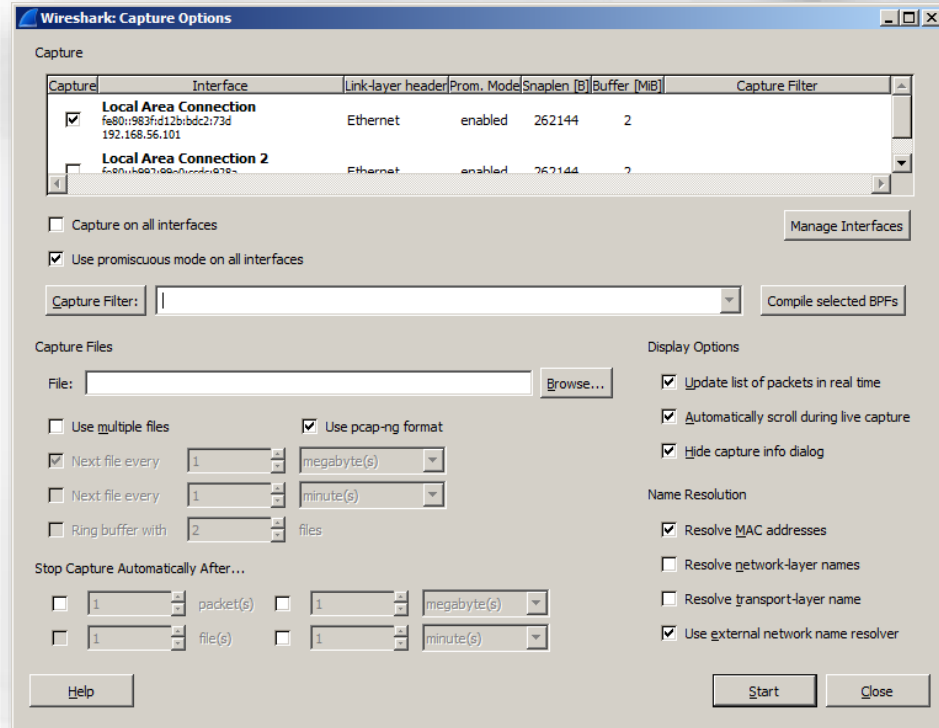
Capture filters will make Wireshark discard packets that do not match the filter. These filters impact how many packets your computer must process and how big the capture file will be.

This is very useful to limit captured traffic in high traffic networks.



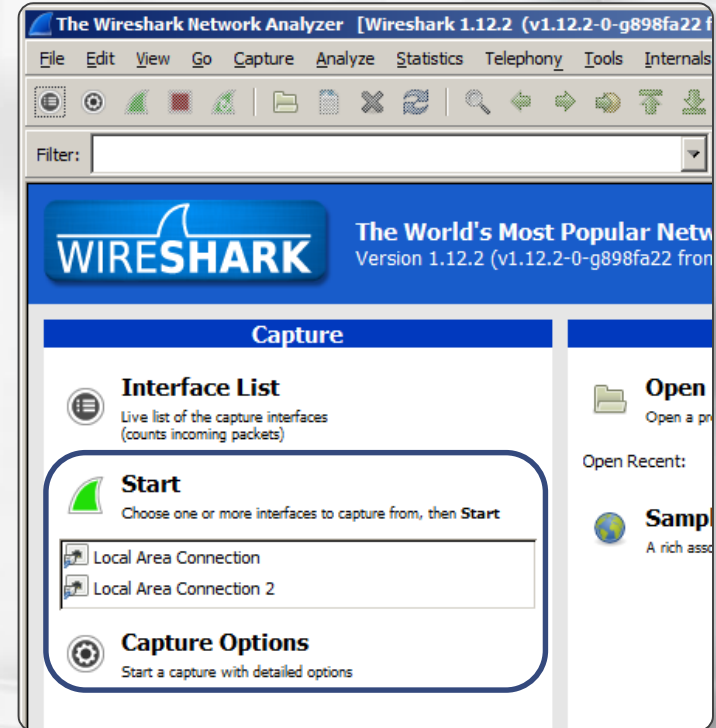
2.8.2 Configuring Wireshark

During your first captures, leave the filter blank and just click **start**.



2.8.2 Configuring Wireshark

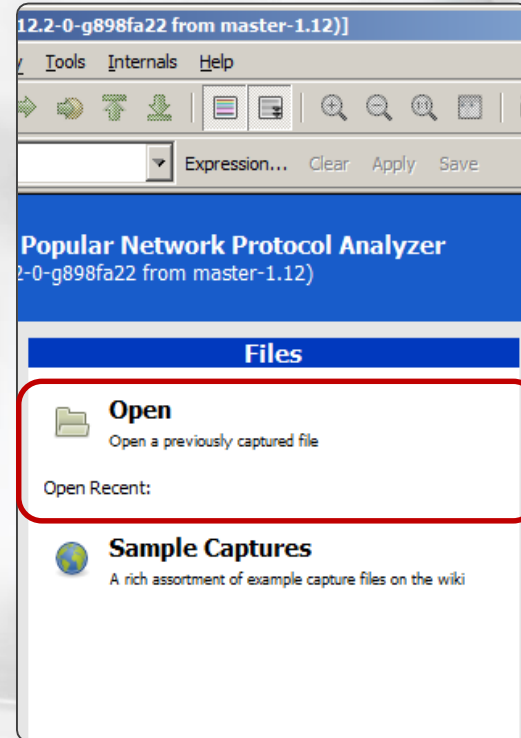
To perform these very same operations, you can just select the capture interface and then click on **capture options** or **start** from the main window.



2.8.2 Configuring Wireshark

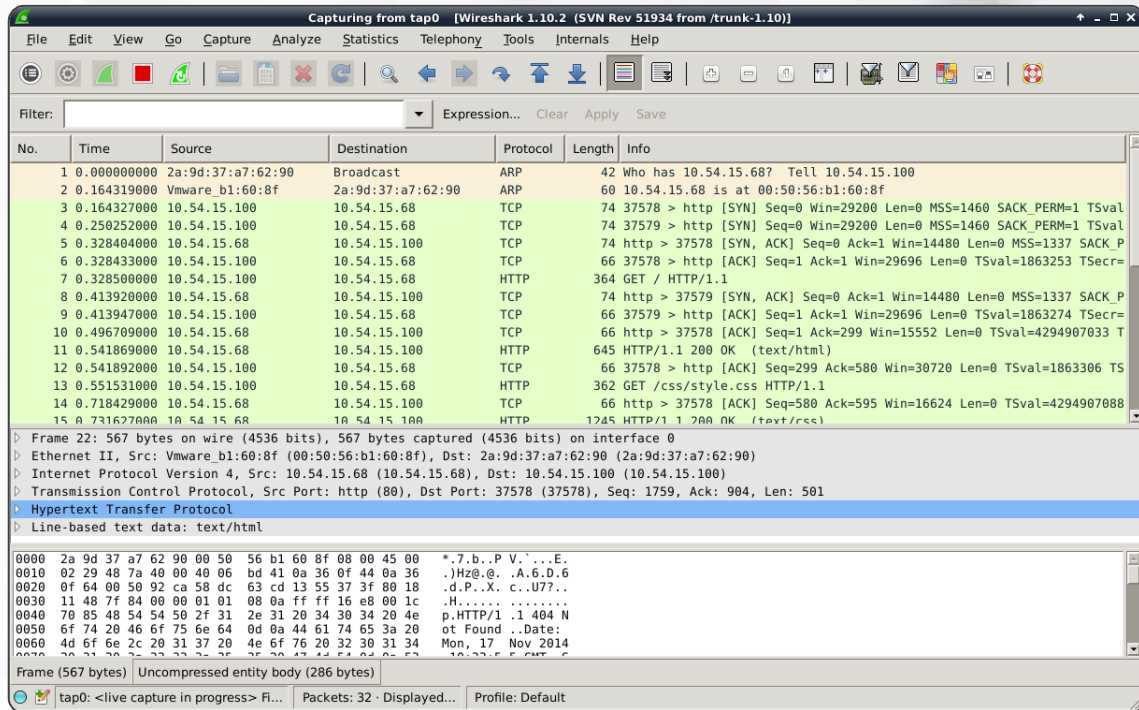
PCAP files store an entire capture (from a previous capture session).

If you already have a PCAP file, you can open it using this button.



2.8.3 The capture window

In either case, doing a live capture or opening a previous one, you will see this interface.



2.8.3 The capture window

The first two columns of the upper pane contain:

- The **number** of the captured packet
- The **arrival time** of the packet in seconds.
The arrival time is relative to the start of the capture.

The image shows the Wireshark network protocol analyzer interface. The title bar indicates the capture is from 'tap0'. The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, and Telephony. The toolbar contains icons for various functions like packet list, packet details, packet bytes, and network statistics. The packet list pane shows 15 packets, with the first packet being a broadcast from 0.0.0.0 to 255.255.255. The packet details pane shows the structure of the Ethernet II and Internet Protocol Version 4 headers.

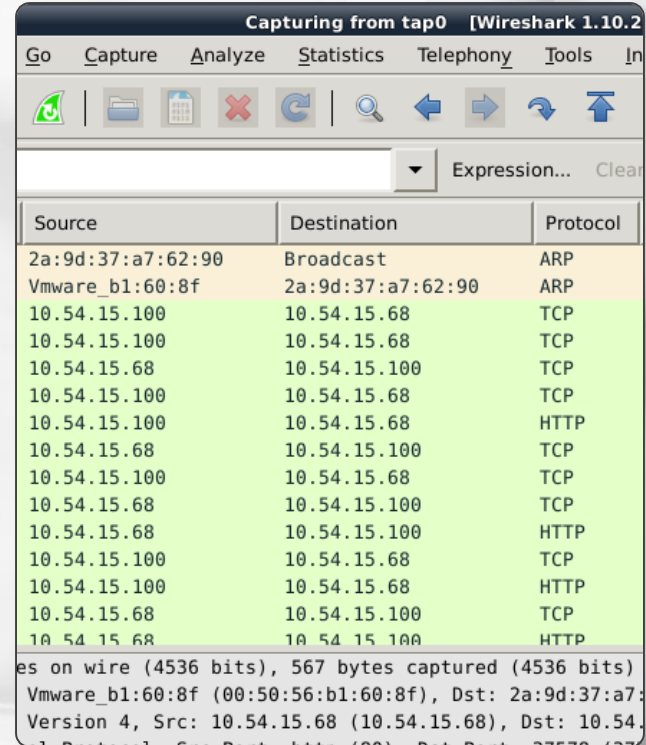
No.	Time	Source	Destination
1	0.000000000	2a:9d:37:a7:62:90	Broadcast
2	0.164319000	Vmware_b1:60:8f	2a:9d:37:a7:62:90
3	0.164327000	10.54.15.100	10.54.15.68
4	0.250252000	10.54.15.100	10.54.15.68
5	0.328404000	10.54.15.68	10.54.15.100
6	0.328433000	10.54.15.100	10.54.15.68
7	0.328500000	10.54.15.100	10.54.15.68
8	0.413920000	10.54.15.68	10.54.15.100
9	0.413947000	10.54.15.100	10.54.15.68
10	0.496709000	10.54.15.68	10.54.15.100
11	0.541869000	10.54.15.68	10.54.15.100
12	0.541892000	10.54.15.100	10.54.15.68
13	0.551531000	10.54.15.100	10.54.15.68
14	0.718429000	10.54.15.68	10.54.15.100
15	0.731627000	10.54.15.68	10.54.15.100

Frame 22: 567 bytes on wire (4536 bits), 567 bytes captured
 Ethernet II, Src: Vmware_b1:60:8f (00:50:56:b1:60:8f), Dst:
 Internet Protocol Version 4, Src: 10.54.15.68 (10.54.15.68)
 Transmission Control Protocol, Seq: 222222222, Win: 65535, Len: 0

2.8.3 The capture window

You can then see the **source**, **destination** and **protocol** columns.

Note how the source and destination address vary according to the protocol.



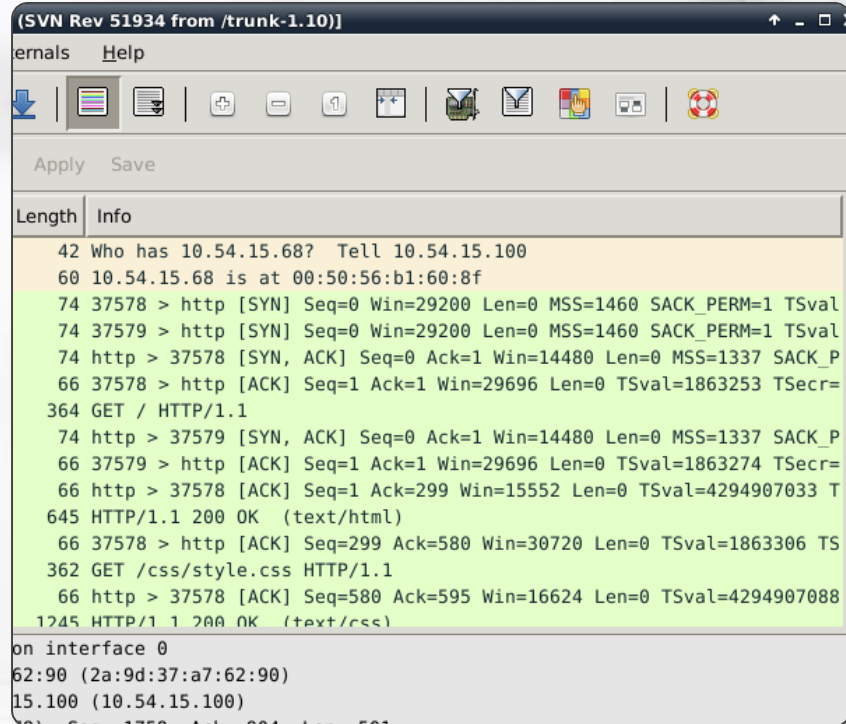
Source	Destination	Protocol
2a:9d:37:a7:62:90	Broadcast	ARP
Vmware_b1:60:8f	2a:9d:37:a7:62:90	ARP
10.54.15.100	10.54.15.68	TCP
10.54.15.100	10.54.15.68	TCP
10.54.15.68	10.54.15.100	TCP
10.54.15.100	10.54.15.68	TCP
10.54.15.100	10.54.15.68	HTTP
10.54.15.68	10.54.15.100	TCP
10.54.15.100	10.54.15.68	TCP
10.54.15.68	10.54.15.100	TCP
10.54.15.68	10.54.15.100	HTTP
10.54.15.100	10.54.15.68	TCP
10.54.15.100	10.54.15.68	HTTP
10.54.15.68	10.54.15.100	TCP
10.54.15.68	10.54.15.100	HTTP

es on wire (4536 bits), 567 bytes captured (4536 bits)
Vmware_b1:60:8f (00:50:56:b1:60:8f), Dst: 2a:9d:37:a7:
Version 4, Src: 10.54.15.68 (10.54.15.68), Dst: 10.54.
1. Protocol: 60:8f (00:50:56:b1:60:8f), Dst: 2a:9d:37:a7:62:90 (2a:9d:37:a7:62:90)

2.8.3 The capture window

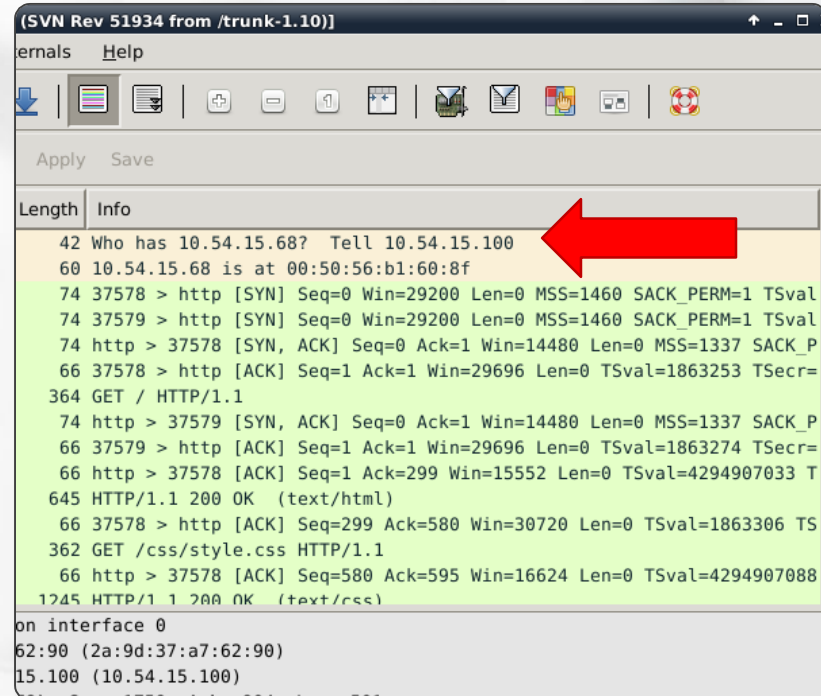
In the last two columns, you can find the **size of the packet** and some related **information**.

The *info* column is protocol specific.



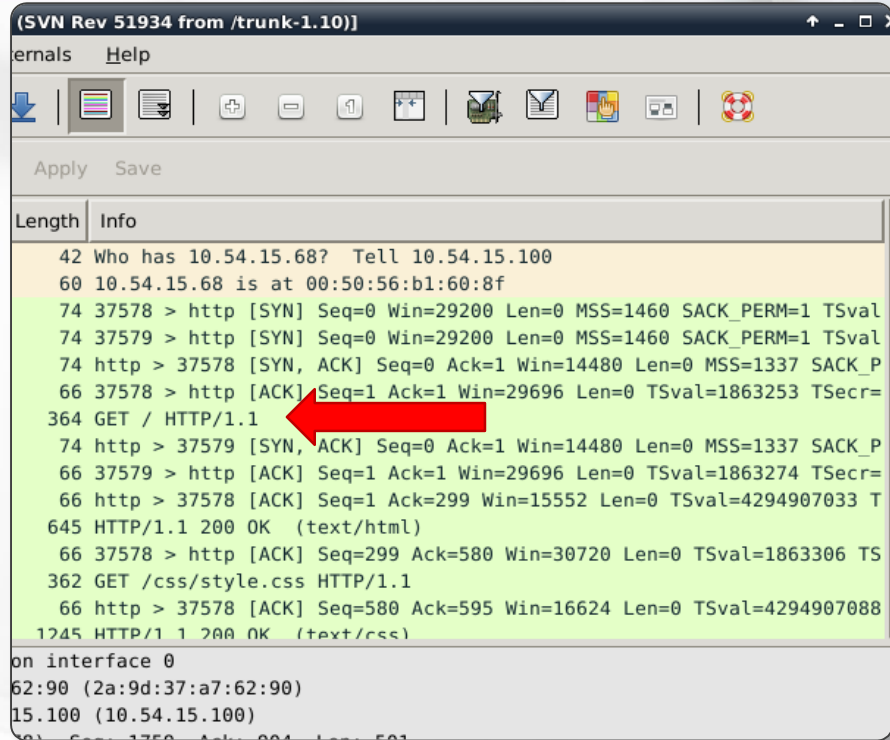
2.8.3 The capture window

For example, the first two packets are ARP requests and replies.



2.8.3 The capture window

This packet is an HTTP request.



The image shows a Wireshark packet capture window titled "(SVN Rev 51934 from /trunk-1.10)". The window displays a list of network packets. The packet list has two columns: "Length" and "Info". The packet at offset 364 is highlighted in green and is an HTTP GET request. A red arrow points to this packet. The packet details pane on the right shows the structure of the packet, including the Ethernet II header, Internet Protocol Version 4 header, and Hypertext Transfer Protocol header. The packet bytes pane at the bottom shows the raw data of the packet.

Length	Info
42	Who has 10.54.15.68? Tell 10.54.15.100
60	10.54.15.68 is at 00:50:56:b1:60:8f
74	37578 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval
74	37579 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval
74	http > 37578 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_P
66	37578 > http [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=1863253 TSecr=
364	GET / HTTP/1.1
74	http > 37579 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1337 SACK_P
66	37579 > http [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=1863274 TSecr=
66	http > 37578 [ACK] Seq=1 Ack=299 Win=15552 Len=0 TSval=4294907033 T
645	HTTP/1.1 200 OK (text/html)
66	37578 > http [ACK] Seq=299 Ack=580 Win=30720 Len=0 TSval=1863306 TS
362	GET /css/style.css HTTP/1.1
66	http > 37578 [ACK] Seq=580 Ack=595 Win=16624 Len=0 TSval=4294907088
1245	HTTP/1.1 200 OK (text/css)

on interface 0
62:90 (2a:9d:37:a7:62:90)
15.100 (10.54.15.100)

2.8.3 The capture window

The center pane gives you access to all the protocol layers used by a packet.

This actually allows you to read the entire packet layer by layer!

```
> Frame 22: 567 bytes on wire (4536 bits), 567 bytes captured (4536 bits) on interface 0
> Ethernet II, Src: Vmware_b1:60:8f (00:50:56:b1:60:8f), Dst: 2a:9d:37:a7:62:90 (2a:9d:37:a7:62:90)
> Internet Protocol Version 4, Src: 10.54.15.68 (10.54.15.68), Dst: 10.54.15.100 (10.54.15.100)
> Transmission Control Protocol, Src Port: http (80), Dst Port: 37578 (37578), Seq: 1759, Ack: 904, Len: 501
> Hypertext Transfer Protocol
> Line-based text data: text/html
```


2.8.3 The capture window

You can drill down to get any information you want about a packet.

For example, this packet has the *ACK* TCP flag on.

```

> Frame 6: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: 2a:9d:37:a7:62:90 (2a:9d:37:a7:62:90), Dst: Vmware_b1:60:8f (00:50:56:b1:60:8f)
> Internet Protocol Version 4, Src: 10.54.15.100 (10.54.15.100), Dst: 10.54.15.68 (10.54.15.68)
> Transmission Control Protocol, Src Port: 37578 (37578), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0
  Source port: 37578 (37578)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 1 (relative sequence number)
  Acknowledgment number: 1 (relative ack number)
  Header length: 32 bytes
  ▾ Flags: 0x010 (ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ....0... = Congestion Window Reduced (CWR): Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
    ....0... = Syn: Not set
    ....0... = Fin: Not set
  Window size value: 29
  [Calculated window size: 29696]
  [Window size scaling factor: 1024]
  > Checksum: 0x30b9 [validation disabled]
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [SEQ/ACK analysis]
```

2.8.3 The capture window

In the bottom pane, you can see the actual packet payload. In this example we see an HTTP GET request.

```
0000 00 50 56 b1 60 8f 2a 9d 37 a7 62 90 08 00 45 00 .PV.`*. 7.b...E.
0010 01 5e 5d b7 40 00 40 06 a8 cf 0a 36 0f 64 0a 36 .^].@.@. ...6.d.6
0020 0f 44 92 ca 00 50 13 55 33 b8 58 dc 5c ef 80 18 .D...P.U 3.X.\...
0030 00 1d 43 de 00 00 01 01 08 0a 00 1c 6e 55 ff ff ..C..... ....nU..
0040 14 6f 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 .oGET / HTTP/1.1
0050 0d 0a 48 6f 73 74 3a 20 31 30 2e 35 34 2e 31 35 ..Host: 10.54.15
0060 2e 36 38 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a .68..Use r-Agent:
0070 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 58 31 Mozilla /5.0 (X1
0080 31 3b 20 4c 69 6e 75 78 20 78 38 36 5f 36 34 3b 1; Linux x86_64;
0090 20 72 76 3a 33 31 2e 30 29 20 47 65 63 6b 6f 2f rv:31.0 ) Gecko/
00a0 32 30 31 30 30 31 30 31 20 46 69 72 65 66 6f 78 20100101 Firefox
00b0 2f 33 31 2e 30 20 49 63 65 77 65 61 73 65 6c 2f /31.0 Ic eweasel/
00c0 33 31 2e 32 2e 30 0d 0a 41 63 63 65 70 74 3a 20 31.2.0.. Accept:
00d0 74 65 78 74 2f 68 74 6d 6c 2c 61 70 70 6c 69 63 text/html,application/xh
00e0 61 74 69 6f 6e 2f 78 68 74 6d 6c 2b 78 6d 6c 2c tml+xml,
00f0 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 6d 6c 3b applicat ion/xml;
0100 71 3d 30 2e 39 2c 2a 2f 2a 3b 71 3d 30 2e 38 0d q=0.9,*/ *;q=0.8.
0110 0a 41 63 63 65 70 74 2d 4c 61 6e 67 75 61 67 65 .Accept- Language
0120 3a 20 65 6e 2d 55 53 2c 65 6e 3b 71 3d 30 2e 35 : en-US, en;q=0.5
0130 0d 0a 41 63 63 65 70 74 2d 45 6e 63 6f 64 69 6e ..Accept -Encodin
0140 67 3a 20 67 7a 69 70 2c 20 64 65 66 6c 61 74 65 g: gzip, deflate
0150 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 ..Connec tion: ke
0160 65 70 2d 61 6c 69 76 65 0d 0a 0d 0a ep-alive ....
```

2.8.4 Filtering

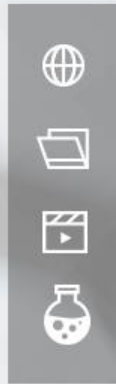
A traffic capture can be overwhelming, even on a network with just a couple of dozens of nodes.

Wireshark can **filter** traffic at **capture** or at **display** time. Each method has its own pros and cons.

```
20 def initialize(context)
21   @context = context
22   @experiment_name = nil
23   @observations = []
24   @control = nil
25   @candidates = []
26   @results = []
27
28   # Initialize the experiment's context
29   per context
30     experiment.context
31   end
32
33   # Initialize the name of the experiment
34   def experiment_name
35     experiment.name
36   end
37
38   # Define the result a match between an observation and a control
39   def matches?
40     # ...
41   end
42
43   # Evaluate the result of the experiment
44   def evaluate_candidates
45     # ...
46   end
47
48   # Evaluate the result of the experiment
49   def evaluate_candidates
50     # ...
51   end
```



You can set capture filters **before** starting the capture so that Wireshark will capture only packets matching the filters.



2.8.4.1 Capture Filters

Here are some basic capture filters.

Syntax	Description
ip	Only packets using IP as layer 3 protocol.
not ip	The opposite of the previous syntax.
tcp port 80	Packets where the source or destination TCP port is 80.
net 192.168.54.0/24	Packets from and to the specified network.
src port 1234	The source port must be 1234; the transport protocol does not matter.
src net 192.168.1.0/24	The source IP address must be in the specified network.
host 192.168.45.65	All the packets from or to the specified host.
host www.examplehost.com	All the packets from or to the specified hostname.

2.8.4.1 Capture Filters

Capture filters will downsize the amount of traffic gathered.

The final capture will be **smaller**, and it will contain **only** the **needed traffic**.



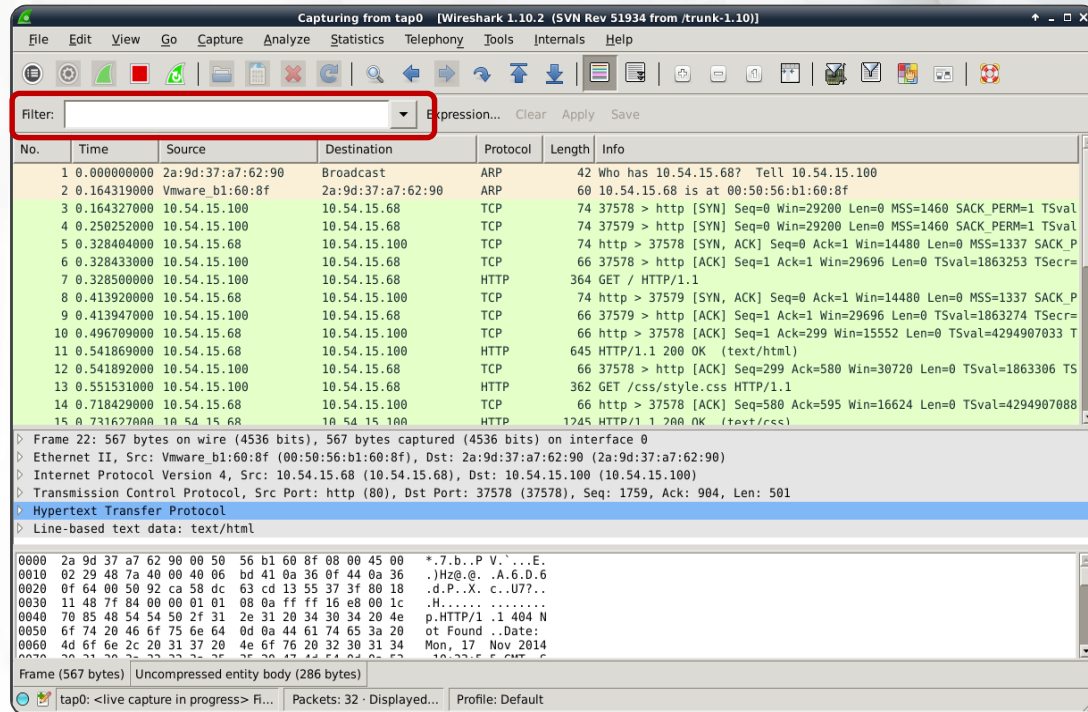
2.8.4.2 Display Filters

However, capture filters might not catch interesting traffic! Display filters instead allow you to inspect and apply very granular filters to every field of the captured packets. Wireshark then displays only the packets matching the filters.

You can always remove or fine tune a display filter, something you can't do with the capture filter (you would have to re-start the capture from scratch).

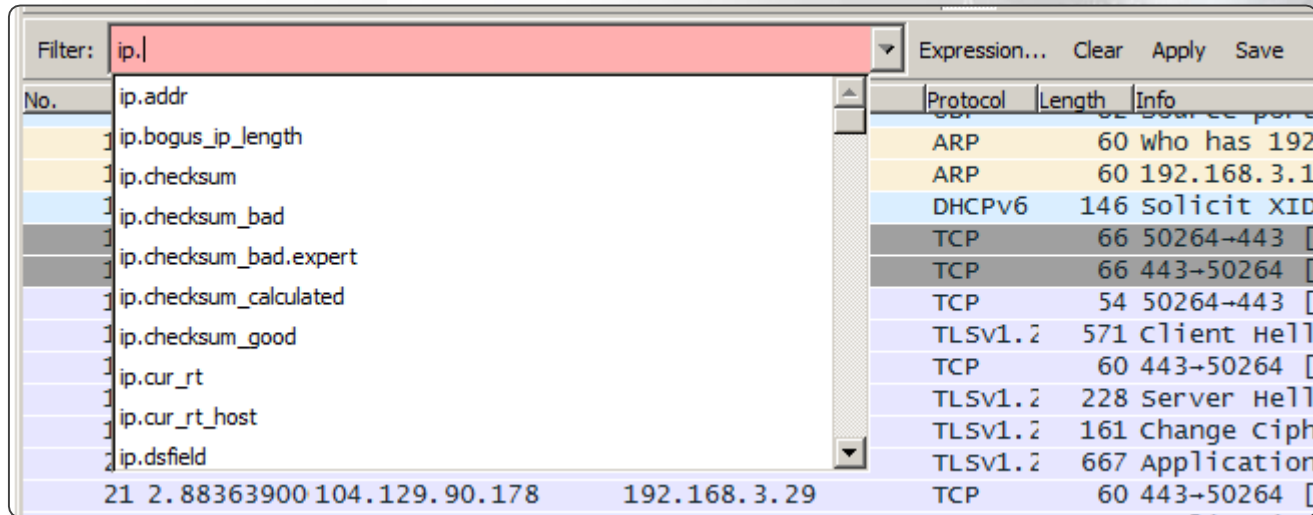
2.8.4.2 Display Filters

You can use the filter textbox to apply a display filter.



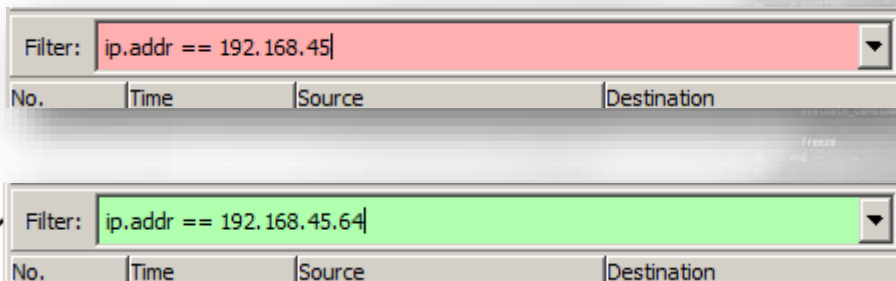
2.8.4.2 Display Filters

You can start typing a filter and Wireshark will give you valid protocol fields.



2.8.4.2 Display Filters

The background of the text-box will turn red if the filter is invalid or green when the filter is valid.



2.8.4.2 Display Filters

A display filter is made by:

Syntax	Description
<Protocolname>	Displays any packet using that protocol.
<Protocolname>[.field]	Displays any packet with the specified field present.
<Protocolname>[.field] [operand value]	Displays any packet whose protocol field matches the operand and value.
<Protocolname>[.field] AND <Protocolname>[.field] [operand value]	You can combine multiple expressions by using logical operators.

2.8.4.2 Display Filters

Below is an example.

Syntax	Description
<code>ip</code>	Displays IP packets.
<code>ip.addr</code>	Displays IP packets with a populated source or destination address.
<code>ip.addr == 192.168.12.13</code>	Displays IP packets with 192.168.12.13 as source or destination address.
<code>ip.addr == 192.168.12.13 or arp</code>	The above or ARP packets.

2.8.4.2 Display Filters

You can find Wireshark display filter reference [here](#).

For any other information, please refer to the [Wireshark User's Guide](#).

2.8.5 Video – Using Wireshark

Using Wireshark

In this video, you will see how to configure and use Wireshark, how capture and display filters work and this powerful tool's main features.



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

2.8.6 Video – Full Stack Analysis with Wireshark

Full Stack Analysis with Wireshark

This video uses the traffic captured in the previous video to perform traffic analysis, which will improve your understanding of low-level interaction between routing, IP, ARP, TCP, application-level protocols and more!



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

2.8.7 Sample Traffic Captures

If you want to practice these topics a little more, you can record some traffic from your computer or you can download a capture from [Wireshark website](http://wiki.wireshark.org/SampleCaptures).

2.8.8 Lab – Data Exfiltration

Apply your fresh knowledge about networking to the Data Exfiltration lab.

You can test it against a environment secured by a firewall, while on the other hand, learn how to bypass this kind of obstacles.

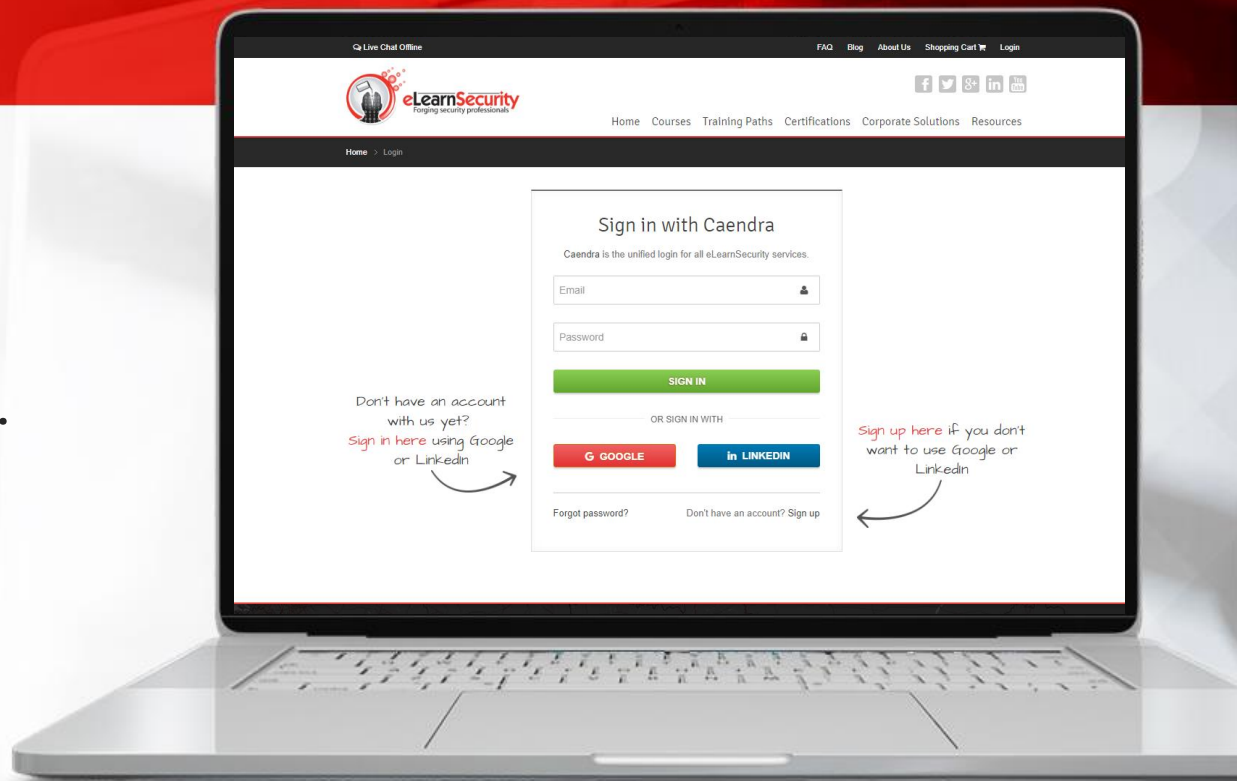


2.8.8 Lab – Data Exfiltration

Data Exfiltration

The environment is secured by a firewall.

Can you bypass it?



**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

References



References

Packet Analysis Reference Guide v3.0

You can download it from the **Resources** drop-down menu of the Networking module (Section Preliminary Skills – Prerequisites)

[RFC 5735 – Special Use IPv4 Addresses](http://tools.ietf.org/html/rfc5735)

<http://tools.ietf.org/html/rfc5735>

[Classful Subnet Calculator](http://www.subnet-calculator.com/)

<http://www.subnet-calculator.com/>

[Service Name and Transport Protocol Port Number Registry](http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml)

<http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>

References



ISO/OSI Model

<http://support.microsoft.com/kb/103884>

Variable Length Subnet Table For IPv4

<http://tools.ietf.org/html/rfc1878>

CIDR Subnet Calculator

<http://www.subnet-calculator.com/cidr.php>

TCPView

<http://technet.microsoft.com/en-us/sysinternals/bb897437>



References

TACO IPv6 Router - a Case Study in Protocol Processor Design

https://www.researchgate.net/publication/31596630_TACO_IPv6_Router_-_a_Case_Study_in_Protocol_Processor_Design

Internet Protocol Version 6 (IPv6) Addressing Architecture

<https://tools.ietf.org/html/rfc3513>

IPv6 Explained for Beginners

<http://www.steves-internet-guide.com/ipv6-guide/>

How to find IPv6 Prefix

<https://networklessons.com/ipv6/how-to-find-ipv6-prefix/>



References

IP Layer Network Administration with Linux

You can download it from the **Resources** drop-down menu of the Networking module (Section Preliminary Skills – Prerequisites)

[The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference](http://www.amazon.com/TCP-Guide-Comprehensive-Illustrated-Protocols/dp/159327047X/ref=sr_1_3?s=books&ie=UTF8&qid=1297880998&sr=1-3)

http://www.amazon.com/TCP-Guide-Comprehensive-Illustrated-Protocols/dp/159327047X/ref=sr_1_3?s=books&ie=UTF8&qid=1297880998&sr=1-3

[Firewall Fundamentals](http://www.amazon.com/Firewall-Fundamentals-Wes-Noonan/dp/1587052210/ref=cm_cr_pr_sims_t)

http://www.amazon.com/Firewall-Fundamentals-Wes-Noonan/dp/1587052210/ref=cm_cr_pr_sims_t

[Wireshark Display Filter Reference](https://www.wireshark.org/docs/dfref/)

<https://www.wireshark.org/docs/dfref/>



References

TCP/IP Tutorial and Technical Overview

You can download it from the **Resources** drop-down menu of the Networking module (Section Preliminary Skills – Prerequisites)

[IPTables tutorial](https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html)

<https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html>

[Wireshark](https://www.wireshark.org/)

<https://www.wireshark.org/>

[Wireshark User's Guide](https://www.wireshark.org/docs/wsug_html_chunked/)

https://www.wireshark.org/docs/wsug_html_chunked/



References



DNS Concepts and Facilities

<https://www.ietf.org/rfc/rfc1034.txt>

Wireshark Sample Captures

<http://wiki.wireshark.org/SampleCaptures>

DNS Implementation and Specification

<https://www.ietf.org/rfc/rfc1035.txt>

IPv6 Calculator

<https://www.ultratools.com/tools/ipv6CIDRToRange>



References

Netfilter

<http://www.netfilter.org/>

Ossec IDS

<http://www.ossec.net/>

FAQ: Network Intrusion Detection Systems

http://www.linuxsecurity.com/resource_files/intrusion_detection/network-intrusion-detection.html



Videos

Using Wireshark

In this video, you will see how to configure and use Wireshark, how capture and display filters work and this powerful tool's main features.

Full Stack Analysis with Wireshark

This video uses the traffic captured in the previous video to perform traffic analysis, which will improve your understanding of low-level interaction between routing, IP, ARP, TCP, application-level protocols and more!

**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

Labs

Find the Secret Server

Use your TCP/IP routing knowledge to access a secret web server.

Data Exfiltration

The environment is secured by a firewall. Can you bypass it?

**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*