



RED ALPHA

Hello World!

Every cybersecurity incident starts with a motivated attacker trying to achieve his goals by attacking a target. The motive can be money, power, fame, or even a desire for revenge. The target can be a person, a company, an organization, or even a country.

In today's world, where almost every electronic device is connected to the Internet Network, this network became one of the most common ways of performing the attack.

Infecting a computer with malware which usually happens throughout the network, and controlling that target computer, is not the goal of the attack but a phase towards the final goal. Moreover, the attacker will usually want to extract information from the infected computer. How can he extract this information? Through the same network which he used to attack the computer.

Therefore, the network plays a significant role in cyber-attacks, and with Network Forensics, one can learn about how the attack was started, what kind of information was leaked out and what is the attacker's motive.

In this lesson, we will meet some network concepts, protocols, and tools that will help us with our investigation. We will also look at a demonstration of a network attack and show how we can investigate it using some existing tools.

Things we will cover:

- [Network packets and protocols](#)
- [Network analyzing](#)
 - [Wireshark](#)
- [Common application protocols](#)
 - [HTTP](#)
 - [DNS](#)
- [Guided example](#)
- [Exercises](#)

Network Packets and Protocols

The data that is being sent over the network can convey different information but is being sent similarly. This is similar to letter mail sending. Getting a letter from the bank and getting a Happy Birthday greeting card, conveys different information but is being sent similarly - using letters with text inside envelopes. The envelopes in the network world are called **network packets** and the type of information they convey is called **application protocol**. As all letters that are sent via the Post Office system need to have a recipient's address and name, similarly, all protocols that will be sent over the computer network will also need to have some addressing in order to reach the correct recipient. These addresses in computer networks are called IP address and MAC address.

IP Address - address in the format of 4 groups of numbers from 0 to 255. Examples: 192.168.1.0, 10.0.5.200, 70.80.1.1. The IP address of a device can be changed, depending on the network it is a member of.

There are special IP addresses that signify that the message is intended for more than one recipient. For example:

- The broadcast IP address is an address that ends with 255, i.e. 10.0.0.255. This means that every device in a network with IP address 10.0.0.X (where X can be any number from 1 to 254) will treat this message as if it was sent to him and will not ignore it.
- The multicast IP address is any address in the range 224.0.0.0 to 224.0.0.255, i.e. 224.0.0.22. This means that a device with the IP 10.0.0.5 can still subscribe to multicast messages and accept any message that reaches him with a multicast IP.

Both broadcast and multicast addresses are considered logical addresses and are not representing real machines.

Lastly, as the number of available IP addresses is limited to about 4 billion, and today there are many more electronic devices that are connected to the internet, we cannot assign a unique IP address for each device. To solve this, there are special IP addresses that are considered private IPs, and they are allowed not to be unique and repeat themselves between different networks. These groups are:

- 10.0.0.0 – 10.255.255.255
- 192.168.0.0 – 192.168.255.255
- 172.16.0.0 – 172.31.255.255

Every IP address which is not in this group is a public IP address, and must be uniquely assigned to a single device. Any device that wants to be accessible through the

Internet network must have a public IP address. If google web server had a private IP address, it would not be accessible via the Internet.

MAC Address - addresses of network cards that cannot be changed as they are burnt in the network card hardware. These addresses are in the following format: XX-XX-XX-XX-XX-XX, where X can be a digit between 0 to 9 or an alphabetical letter between A to F. 00-1A-A0-52-76-9F is an example of a valid MAC address. The first 3 groups of the MAC address uniquely identify who is the manufacturer of this network device, and they are called OUI (Organization Unit Id). All of the network cards manufactured by TP-LINK will start with D0-37-45. This information can be found online by looking for "TP-LINK MAC address OUI".

Similar to IP, MAC addresses also have special addresses: Broadcast and Multicast:

- Broadcast MAC address is always the address FF-FF-FF-FF-FF-FF. This means that every device that gets a packet with destination MAC which is equal to FF-FF-FF-FF-FF-FF, will not ignore this message.
- Multicast MAC addresses are addresses that start with a special prefix. Examples are: 01-80-C2-XX-XX-XX, 01-00-5E-XX-XX-XX, 33-33-XX-XX-XX-XX

But with these two addresses, IP and MAC, a network packet might be able to reach my computer but my computer won't know to which of all the running applications this packet is relevant for. In order to solve this, we need another kind of address called a **port number**. Port number - A number between 1 - 65,535, which identifies a specific running application that listens for incoming network packets.

All of the network packets that you will capture, although they might convey different kinds of information and as such will be of different **application protocols**, they will all contain source and destination IP address, MAC address, and Port number, which will tell us the addresses of the sender and the recipient respectively.

Examples of known and widely used application protocols are

- HTTP (Hypertext Transfer Protocol)
 - Generic and can be used for many purposes
 - Get HTML web pages
 - Download / Upload files
 - Video streaming
 - Client / Server based - one side is the client which initiates the HTTP request, while another side is the server which can only respond to client requests

- HTTP servers are addressable at port 80 or 443 for a secured version of HTTP called HTTPS where the client request and server response are both encrypted and can be decrypted only by them
- DNS (Domain Name System)
 - As it is hard for humans to remember IP address, DNS is used to translate domain names to IP address, such as www.google.com → 72.17.11.228
 - DNS is always used behind the scenes when you browse the internet or play online games where your game client tries to access the game server

We will dive more in-depth into these two protocols later on.

Network Analyzing

What Is A Network Packet?



```

0000: 02 40 00 D8 03 02 53 43 5B 90 9D 9B 72 0B BC 0C .@...SC[...r...
0010: BC 2B 92 A8 48 97 CF BD 39 04 CC 16 0A 85 03 90 .+..H...9.....
0020: 9F 77 04 33 D4 DE 00 00 66 C0 14 C0 0A C0 22 C0 .w.3....f.....
0030: 21 00 39 00 38 00 88 00 87 C0 0F C0 05 00 35 00 !.9.8.....5..
0040: 84 C0 12 C0 08 C0 1C C0 1B 00 16 00 13 C0 0D C0 .....
0050: 03 00 0A C0 13 C0 09 C0 1F C0 1E 00 33 00 32 00 .....3.2.
0060: 9A 00 99 00 45 00 44 C0 0E C0 04 00 2F 00 96 00 ....E.D..../.
0070: 41 C0 11 C0 07 C0 0C C0 02 00 05 00 04 00 15 00 A.....
0080: 12 00 09 00 14 00 11 00 08 00 06 00 03 00 FF 01 .....
0090: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4.
00a0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2.....
00b0: 0A 00 16 00 17 00 08 00 06 00 07 00 14 00 15 00 .....
00c0: 04 00 05 00 12 00 13 00 01 00 02 00 03 00 0F 00 .....
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 2C 2A 2F 2A ...#.....*/
00e0: 3B 71 3D 30 2E 38 0D 0A 41 63 63 65 70 74 2D 4C ;q=0.8..Accept-L
00f0: 61 6E 67 75 61 67 65 3A 20 65 6E 2D 55 53 2C 65 uage: en-US,e
0100: 6E 3B 71 3D 30 2E 35 0D 0A 41 63 63 65 70 74 2D n;q=0.5..Accept-
0110: 45 6E 63 6F 64 69 6E 67 3A 20 67 7A 69 70 2C 20 Encoding: gzip,
  
```

Here is an example of a capture of a network packet. It might be easy to look at the textual parts of the packet (the red square) and try to understand what protocol this packet encapsulates. In this example, we can see known HTTP headers like “Accept-Language” and “Accept-Encoding” and conclude that this is an HTTP packet. However, not all protocols are textual, and in this example, it might be cumbersome to identify what is the source/destination IP of this packet, or the source/destination port. For this, we have some analysis tools that can help us to dissect the protocols. We will cover one of the most common and easy to use, called Wireshark

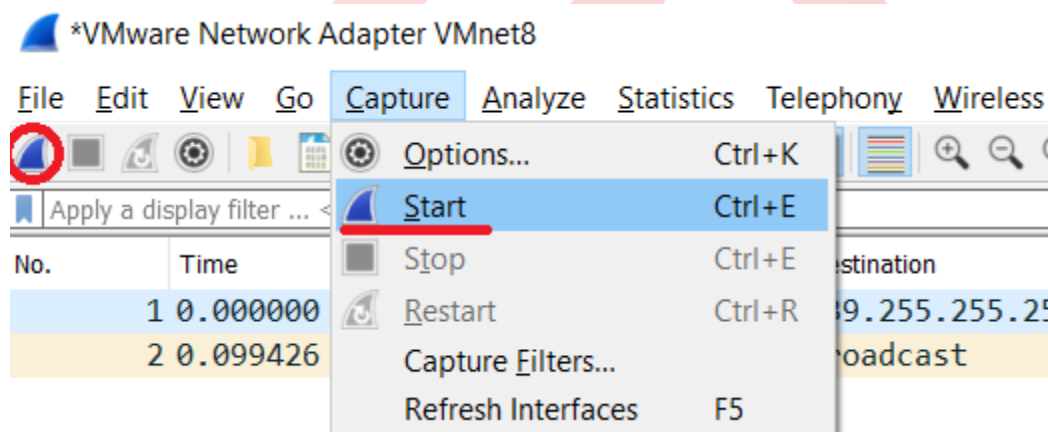
Wireshark

Wireshark is like the guard at the entrance of a condominium. As the guard logs in a logbook every person who enters or leaves the condo, Wireshark logs every incoming data that reaches a computer's network card, or outgoing data that leaves it. This will include any malicious network activity. It allows us not only to monitor network traffic but also to analyze it. The process of monitoring the network is called **sniffing**. Wireshark can be used to **sniff** the traffic and analyze it in real-time, or load an existing sniff file, called a *Packet Capture* (**.PCAP**) file, and analyze it offline.

This tool is extremely useful for learning hands-on networking and understanding how everything works. Want to know how your browser communicates with google? Or how does your torrent client download gigabytes of movies from thousands of peers? Open Wireshark and start analyzing!

Start with downloading Wireshark [here](#). Once the download is finished, install it.

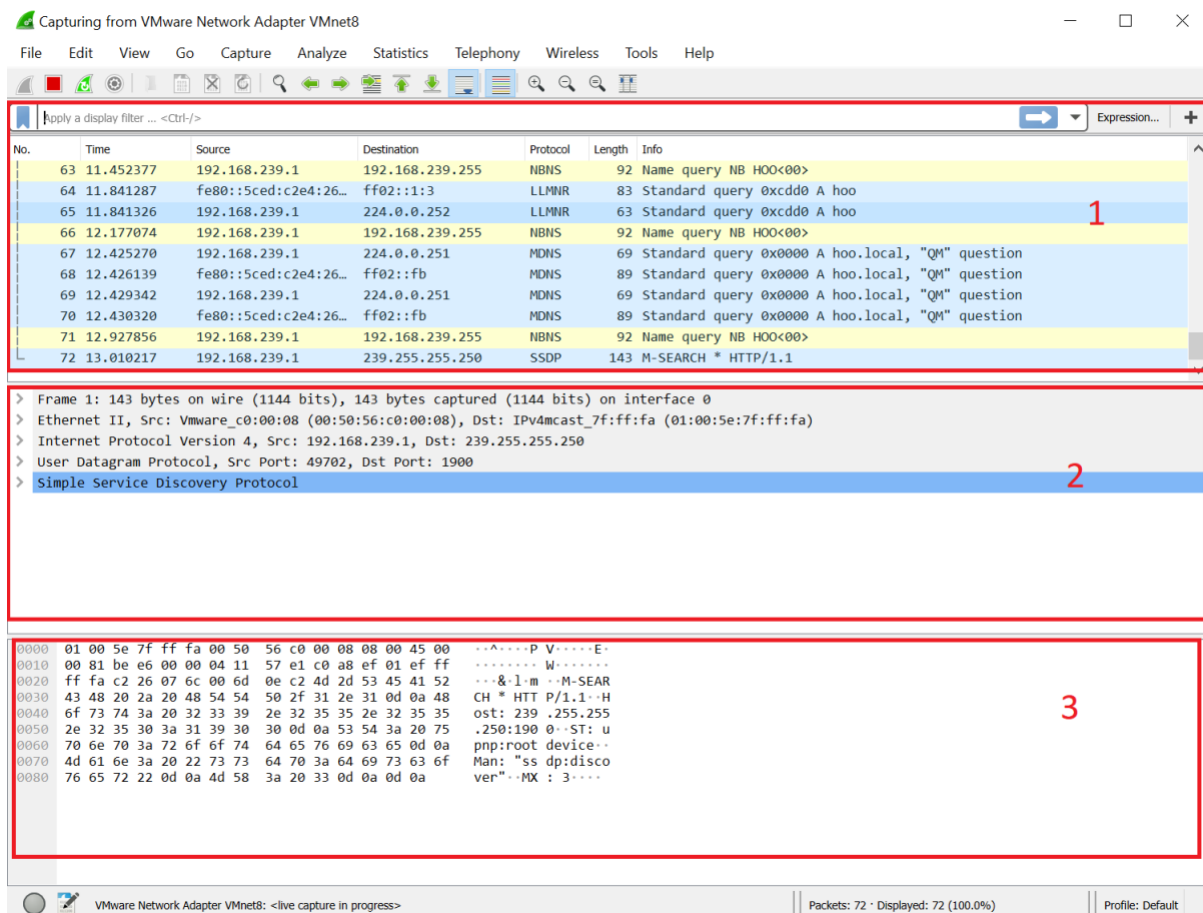
Open up Wireshark. Go to **Capture | Start** or click the fin icon to start a new capture.



Open up your browser and go to your favorite search engine, bing. You will notice Wireshark is recording a lot of traffic. Do not worry, by the end of this tutorial, you will understand almost all of it.

Wireshark is smart and can give us **hints** on what kind of data we are dealing with. It does so by dissecting the frames and making an educated guess of the protocols it includes. It does a great job, but remember that in the end everything is just **raw data** and it can mean different things according to the context.

Let's take a closer look at the Wireshark interface:



There are 3 different panes that are open in Wireshark:

1. The first pane includes all the captured frames. By default, they are sorted by time, but we can sort them however we like. We can click on a packet and see more information in the other 2 panes.
2. The second pane shows a breakdown of the frame into protocols. We can see the frame consists of multiple protocols which encapsulate the ones below them - Ethernet (Data Link), IPv4 (Network), TCP (Transport), and HTTP (Application).
3. The third layer is a binary dump of the data. On the right, we can see a textual representation of the data, and on the left, we can see the hex view of the raw frame data. Hex is a shorter way to represent binary data, where 4 bits are converted to a single alphanumeric character, from 0 to F.

We can filter out frames by using the display filter. Since Wireshark understands protocols and automatically figures them out, we can use this to our advantage. Let's filter by source IP address, using the destination IP from the above frame.

ip.dst==192.168.1.3						
No.	Time	Source	Destination	Protocol	Length	Info
2	0.094573	172.217.23.174	192.168.1.3	TCP	54	443 → 55273 [ACK] S...
3	0.102109	172.217.23.174	192.168.1.3	TLSv1.2	132	Application Data
4	0.103135	172.217.23.174	192.168.1.3	TLSv1.2	92	Application Data
6	0.104257	172.217.23.174	192.168.1.3	TLSv1.2	100	Application Data
10	0.199019	172.217.23.174	192.168.1.3	TCP	54	443 → 55274 [ACK] S...
11	0.200023	172.217.23.174	192.168.1.3	TCP	54	443 → 55274 [ACK] S...
12	0.210732	172.217.23.174	192.168.1.3	TLSv1.2	292	Application Data

Using a filter like `ip.dst == 192.168.1.3`, Wireshark now only shows frames that have the required destination IP. We can combine filters using the logical operators **and/or**:

ip.dst == 104.193.88.77 and http.request.method == "GET"						
No.	Time	Source	Destination	Protocol	Length	Info
7	3.302606	172.17.249.119	104.193.88.77	HTTP	661	GET / HTTP/1.1
105	4.013911	172.17.249.119	104.193.88.77	HTTP	802	GET /img/PCtm_d

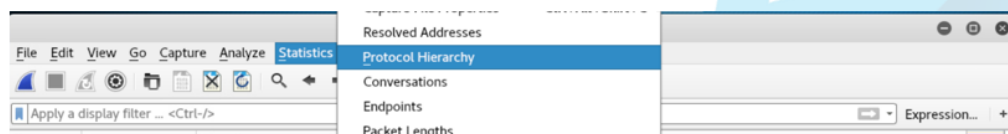
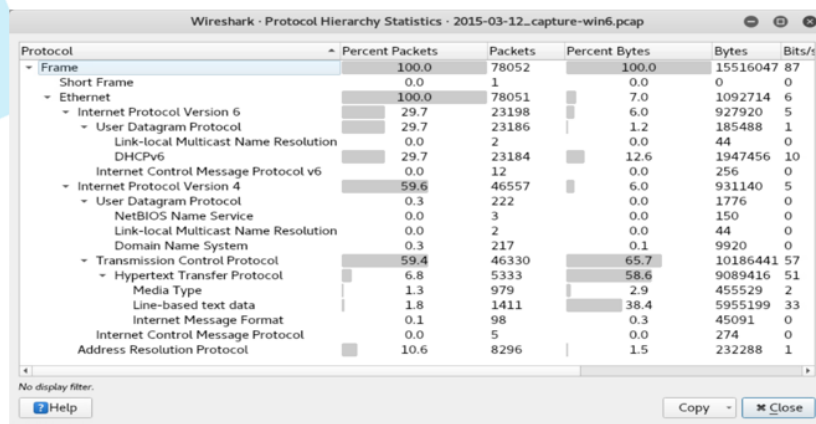
We can also filter frames that contain the IP protocol by providing the IP filter without any other info.

ip				
No.	Time	Source	Destination	Protocol
1	0.000000	192.168.1.3	172.217.23.174	TLSv1
2	0.094573	172.217.23.174	192.168.1.3	TCP
3	0.102109	172.217.23.174	192.168.1.3	TLSv1
4	0.103135	172.217.23.174	192.168.1.3	TLSv1
5	0.103222	192.168.1.3	172.217.23.174	TCP
6	0.104257	172.217.23.174	192.168.1.3	TLSv1
7	0.104571	192.168.1.3	172.217.23.174	TLSv1

> Frame 1: 161 bytes on wire (1288 bits), 161 bytes captured (1288 bits) on interface 0
 > Ethernet II, Src: EdimaxTe_a1:ab:1c (80:1f:02:a1:ab:1c), Dst: D-Link_80:00:00:00:00:00
 > Internet Protocol Version 4, Src: 192.168.1.3, Dst: 172.217.23.174
 > Transmission Control Protocol, Src Port: 55273, Dst Port: 443, Seq: 3044128000
 > Transport Layer Security

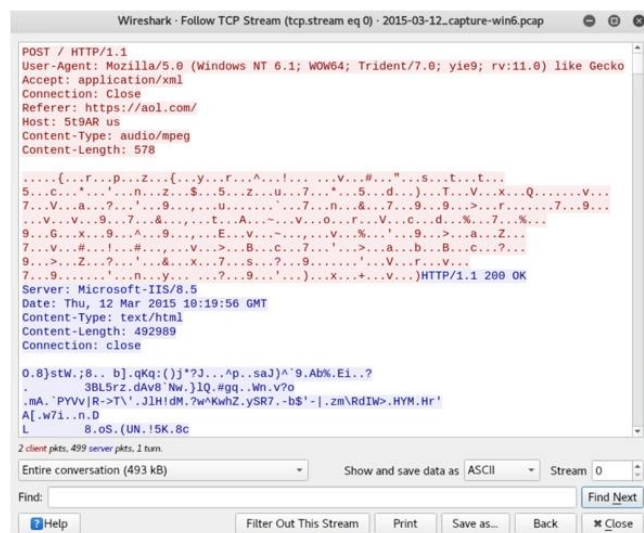
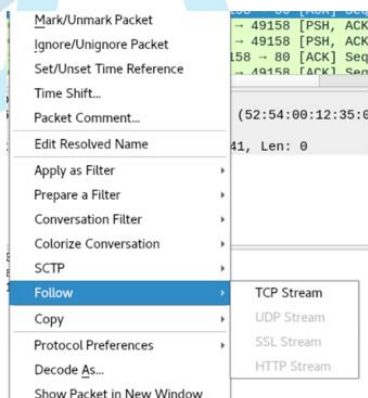
As an analysis tool, another thing we get from Wireshark is statistical data. Protocol Hierarchy is an example of it. We can see visually what is the amount of traffic being sent or received per protocol, and identify the most common protocols in our capture.

Protocol Hierarchy



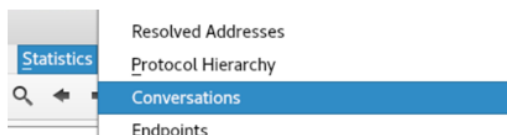
Usually, application data is being sent in multiple TCP segments. What we are interested in, is in the application data itself and not the different TCP segments with their fields. “Follow TCP Stream” is a very useful feature in Wireshark which builds for us all of the TCP segments of a specific TCP connection that we choose, and shows us only the application data that was transmitted as a single stream.

Follow TCP Stream



Conversations and Endpoints are another two types of statistical data. “Conversations” shows packet exchange between two entities in the network while “Endpoints” shows a summary of all communication sent to or from a specific entity.

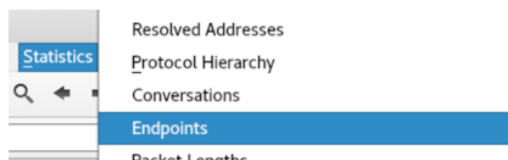
Conversations



The screenshot shows the 'Wireshark - Conversations - 2015-03-12_capture-win6.pcap' window. It displays a table of network conversations. The table has columns for Address A, Port A, Address B, Port B, Packets, Bytes, Packets A → B, Bytes A → B, Packets B → A, Bytes B → A, and Re. The table lists several conversations between 10.0.2.106 and various IP addresses, including 49158, 49159, 49160, 49161, 49162, 49163, 49164, 49165, 49166, 49167, 49168, 49169, 49170, 49171, 49172, and 49173. The table also shows the number of packets and bytes for each conversation, as well as the direction of the traffic (A → B or B → A).

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Re
10.0.2.106	49158	95.163.121.33	80	659	529 k	155	9,201	504	520 k	18
10.0.2.106	49159	115.244.227.73	80	3	194	3	194	0	0	24
10.0.2.106	49160	188.165.26.237	80	3	194	3	194	0	0	25
10.0.2.106	49161	183.99.131.141	80	12	1,660	6	952	6	708	26
10.0.2.106	49162	85.253.42.5	80	3	194	3	194	0	0	27
10.0.2.106	49163	87.236.215.105	80	10	1,208	5	554	5	654	27
10.0.2.106	49164	87.236.215.105	80	12	2,327	6	1,313	6	1,014	27
10.0.2.106	49165	87.236.215.105	80	12	1,835	6	933	6	902	27
10.0.2.106	49166	87.236.215.105	80	29	16 k	11	1,414	18	14 k	27
10.0.2.106	49167	1.164.108.76	80	3	194	3	194	0	0	28
10.0.2.106	49168	87.236.215.105	80	148	140 k	28	2,245	120	138 k	28
10.0.2.106	49169	87.236.215.105	80	12	2,078	6	1,235	6	843	28
10.0.2.106	49170	87.236.215.105	80	51	44 k	13	1,364	38	42 k	28
10.0.2.106	49171	87.236.215.105	80	12	1,935	6	1,097	6	838	30
10.0.2.106	49172	87.236.215.105	80	12	1,844	6	942	6	902	30
10.0.2.106	49173	87.236.215.105	80	12	1,977	6	1,139	6	838	30

Endpoint Statistics

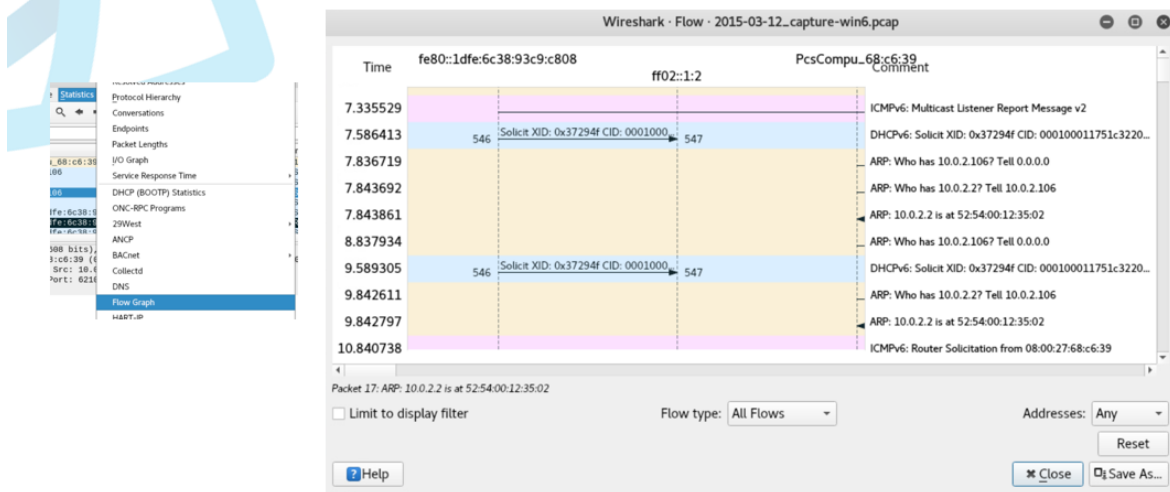


The screenshot shows the 'Wireshark - Endpoints - 2015-03-12_capture-win6.pcap' window. It displays a table of endpoint statistics. The table has columns for Address, Packets, Bytes, Tx Packets, Tx Bytes, Rx Packets, Rx Bytes, Country, City, AS Number, and AS Organization. The table lists statistics for various IP addresses, including 1.163.134.60, 1.163.153.38, 1.163.153.230, 1.164.99.244, 1.164.108.76, 1.164.113.124, 1.164.118.4, 1.164.178.101, 1.164.181.16, 1.186.166.248, 2.194.50.10, 2.194.69.231, 2.194.79.27, 5.101.97.66, 5.135.28.104, and 8.8.4.4. The table also shows the number of packets and bytes for each endpoint, as well as the direction of the traffic (Tx or Rx).

Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	AS Number	AS Organization
1.163.134.60	1,089	319 k	612	250 k	477	68 k	—	—	—	—
1.163.153.38	15	970	0	0	15	970	—	—	—	—
1.163.153.230	3	194	0	0	3	194	—	—	—	—
1.164.99.244	312	39 k	132	13 k	180	25 k	—	—	—	—
1.164.108.76	3	194	0	0	3	194	—	—	—	—
1.164.113.124	118	19 k	59	8,838	59	10 k	—	—	—	—
1.164.118.4	487	78 k	244	35 k	243	42 k	—	—	—	—
1.164.178.101	71	9,319	32	3,184	39	6,135	—	—	—	—
1.164.181.16	47	6,229	22	2,256	25	3,973	—	—	—	—
1.186.166.248	3	194	0	0	3	194	—	—	—	—
2.194.50.10	111	16 k	55	6,729	56	9,726	—	—	—	—
2.194.69.231	15	970	0	0	15	970	—	—	—	—
2.194.79.27	15	970	0	0	15	970	—	—	—	—
5.101.97.66	261	24 k	75	6,163	186	17 k	—	—	—	—
5.135.28.104	11,546	2,322 k	6,281	1,812 k	5,265	510 k	—	—	—	—
8.8.4.4	4	348	2	196	2	152	—	—	—	—

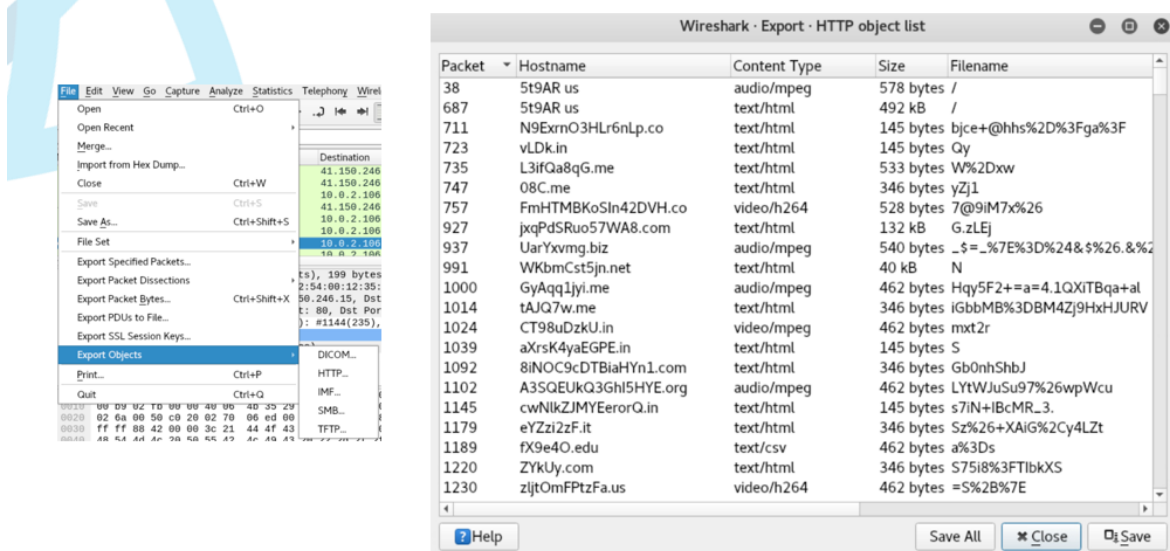
Flow graph provides us with time-based statistics about the flow of communication. It helps us to see what was the chronological order of the communication. We might expect to see a DNS query before an HTTP request to a specific web server.

Flow Graph



Many times the way in which malware enters a system is by downloading it from a malicious web server using the HTTP protocol. Export HTTP objects allow us to save resources that were downloaded using the HTTP protocol to the disk and research them in a sandbox environment.

Export HTTP Objects



Common application protocols

HTTP

HTTP is a generic protocol and can be used for many purposes which made it so popular and widely used.

This document is protected by copyright laws and contains material proprietary to Red Alpha Cybersecurity Pre. Ltd. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Red Alpha Cybersecurity. The receipt or possession of this document does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

It is a Client / Server based protocol which means that there will always be a client sending an **HTTP request** to a server which in return will send an **HTTP response**.

Imagine one computer in a network wants to download a file from another computer in the network, you can run the second one as an HTTP server and in the first one, use your web browser as the HTTP client that will send an HTTP request asking for the desired file. The server then will send an HTTP response containing the file.

Another example can be a smart light bulb that acts as an HTTP server and using your web browser or a mobile app, you will be able to turn on and off the lights via HTTP requests.

HTTP Request

HTTP requests contain information that specifies the type of the request and the desired resource or command we are asking from the server. Here is how an HTTP request looks like:



Let's explain some of these fields:

URI

URI

- An identifier to any resource on the server

```
scheme "://" host [ ":" port ] [ abs_path [ "?"  
query ] ]  
"https://cscc.edu/courses/python/exercises?sort_by=  
grade"  
"http://192.168.1.50:8080/test?to=Bob&message=hi"
```

A uniform resource identifier (**URI**) identifies a resource. URIs are generic and may be used for various purposes, like pointing to files

The HTTP URI consists of 5 main parts:

- scheme - The protocol used, either **http** or **https**
- host - the host running the webserver
- port - port of the webserver. If not specified, the default port for the protocol is used (80 for http, 443 for https)
- path - Path to the resource on the server
- query - appears at the end of the path after the question mark. This is optional and allows passing parameters to the webserver. Parameters are given key value pairs and are separated using the and (&) sign. For example, Google uses the **q** parameter for its query string. Try this: <https://www.google.com/search?q=singapore>

HTTP Methods

GET and POST are the most common methods that are implemented by all servers and are utilized by your web browser directly

- GET - when you browse google, your browser uses GET requests to retrieve the content of the page
- POST - When you send a form like a signup form, the browser sends the form parameters through the HTTP body in the same syntax of a query. This is useful because the URI has a limited length. This may also be used for sending large parameters like files.

HTTP Headers

Lastly, an HTTP request allows the client to pass additional information using **headers**, which are key and value pairs like words and their respective translations in a dictionary.

We always use headers when we want to add additional information that is not related to the data, but for the communication.

Let's take the "User Agent" header as an example. The "User Agent" is a string that conveys information about the requesting client such as:

- application (chrome / firefox / safari / etc.)
- operating system (Windows / MacOS / Android / etc.)
- Browser engine version (AppleWebKit 525.13)

This header allows the server to respond with different content (or style) for different clients. For example, if a client is accessing the server from a mobile phone, the server can respond with a mobile-friendly version of the web page which is different from the response that a client who is surfing from a desktop computer will receive.

Example of a User Agent header value:

Mozilla/5.0 (Windows NT 6.0; WOW64) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.45 Safari/535.19



Luckily, some online tools can extract the information from a User Agent string and explain it better:

User Agent String explained :

```
Mozilla/5.0 (Windows NT 6.0; WOW64) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.45 Safari/535.19
```

Copy/paste any user agent string in this field and click 'Analyze'

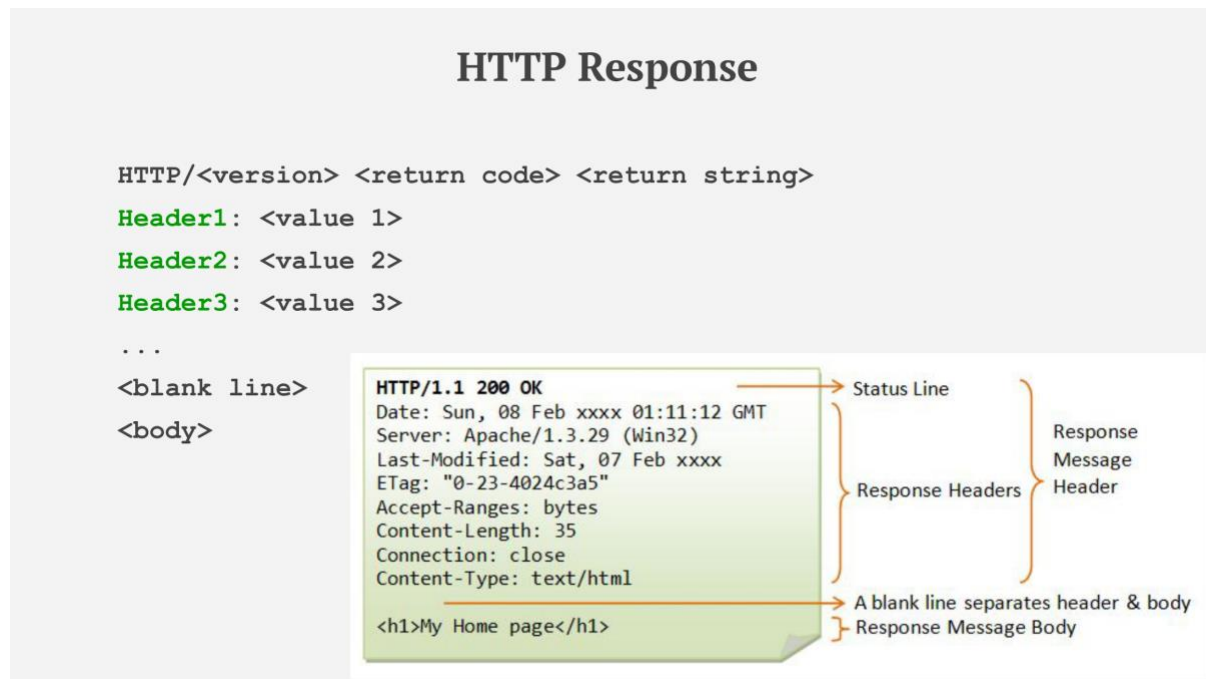
Analyze

Chrome 18.0.1025.45	
Mozilla	MozillaProductSlice. Claims to be a Mozilla based user agent, which is only true for Gecko browsers like Firefox and Netscape. For all other user agents it means 'Mozilla-compatible'. In modern browsers, this is only used for historical reasons. It has no real meaning anymore
5.0	Mozilla version
Windows NT 6.0	Operating System:  Windows Vista
WOW64	(Windows-On-Windows 64-bit) A 32-bit application is running on a 64-bit processor
AppleWebKit	The Web Kit provides a set of core classes to display web content in windows
535.19	Web Kit build
KHTML	Open Source HTML layout engine developed by the KDE project
like Gecko	like Gecko...
Chrome	Name :  Chrome
18.0.1025.45	Chrome version
Safari	Based on Safari
535.19	Safari build

This information can be very valuable when analyzing HTTP network traffic as we can learn about the actual devices in the communications, including their OS version and whether they are using a desktop or a mobile device.

HTTP Response

The HTTP response contains information that specifies whether the request was handled successfully, the type and size of the response content, and the requested resource itself. Here is how an HTTP request looks like:



HTTP Response Codes

Indicate whether a specific HTTP request has been completed successfully. Responses are grouped into five classes:

- Successful responses (200–299) – i.e. 200 OK
 - The server successfully handled the request
- Redirects (300–399) – i.e. 301 Moved Permanently
 - The requested resource has been moved somewhere else, the client should make a new request to the new URI
- Client errors (400–499) – i.e. 400 Bad Request / 404 Not Found
 - The client is sending a malformed request, doesn't have permission to access the resource, or the resource he is trying to access does not exist.
This usually indicates a failure due to the client's fault
- Server errors (500–599) – i.e. 500 Internal Server Error
 - An error on the server side prevented it from handling the client request.
This usually indicates a failure due to the server's fault

HTTP in Wireshark

Most HTTP traffic is secured using HTTPS which cannot be understood using Wireshark.

However, there are still some large websites that use HTTP:

Alexa Rank	Website
4.	 baidu.com
6.	 qq.com
14.	 sohu.com
15.	 jd.com
20.	 sina.com.cn
35.	 t.co
48.	 wikia.com

Let's examine the traffic generated when browsing baidu.com, searching for kittens. We can see the request methods, user agent, and parameters:

```
172.21.104.252 45.113.192.102 HTTP 1704 GET /s?ie=utf-8&mod=1&isbd=1&isid=F8084990F2A80707&ie=utf-8
45.113.192.102 172.21.104.252 HTTP 74 HTTP/1.1 200 OK (text/html)

> Frame 67: 1704 bytes on wire (13632 bits), 1704 bytes captured (13632 bytes) on interface 0
> Ethernet II, Src: Microsoft Wi-Fi [88:E7:2F:EE:8C:29], Dst: Microsoft Wi-Fi [88:E7:2F:EE:8C:29]
> Internet Protocol Version 4, Src: 172.21.104.252, Dst: 45.113.192.102
> Transmission Control Protocol, Src Port: 50096, Dst Port: 80, Seq: 1, Len: 1704
< Hypertext Transfer Protocol
  < [truncated]GET /s?ie=utf-8&mod=1&isbd=1&isid=F8084990F2A80707&ie=utf-8 HTTP/1.1
    < [truncated]Expert Info (Chat/Sequence): GET /s?ie=utf-8&mod=1&isbd=1&isid=F8084990F2A80707&ie=utf-8
      Request Method: GET
    < Request URI [truncated]: /s?ie=utf-8&mod=1&isbd=1&isid=F8084990F2A80707&ie=utf-8
      Request URI Path: /s
    < Request URI Query [truncated]: ie=utf-8&mod=1&isbd=1&isid=F8084990F2A80707&ie=utf-8
      Request URI Query Parameter: ie=utf-8
      Request URI Query Parameter: mod=1
      Request URI Query Parameter: isbd=1
      Request URI Query Parameter: isid=F8084990F2A80707
      Request URI Query Parameter: ie=utf-8
      Request URI Query Parameter: f=8
      Request URI Query Parameter: rsv_bp=1
      Request URI Query Parameter: rsv_idx=1
      Request URI Query Parameter: tn=baidu
      Request URI Query Parameter: wd=kittens
    < User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
    < Cache-Control: max-age=0
    < Accept: */*
    < Accept-Language: en-US,en;q=0.5
    < is_referer: http://www.baidu.com/s?ie=utf-8&f=8&rsv_bp=1&rsv_idx=1&tn=baidu&wd=puppies&ie=utf-8
    < is_xhr: 1
    < X-Requested-With: XMLHttpRequest
    < Accept-Encoding: gzip, deflate
    < Host: www.baidu.com
    < Connection: Keep-Alive
```

For the HTTP response, we can see interesting headers:

- Which webserver is returning our answers
- Content-Type - distinguishes different file types, like scripts, binary, or plain text files.
- Content-Encoding - compression method

```

45.113.192.102    172.21.104.252    HTTP    74 HTTP/1.1 200 OK (text/html)
> Frame 140: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on in
> Ethernet II, Src: Microsof_81:ea:42 (00:15:5d:81:ea:42), Dst: Microsof_7e:
> Internet Protocol Version 4, Src: 45.113.192.102, Dst: 172.21.104.252
> Transmission Control Protocol, Src Port: 80, Dst Port: 50096, Seq: 39906,
> [36 Reassembled TCP Segments (39925 bytes): #70(1442), #72(769), #74(1460)
▼ Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Bdpagetype: 3\r\n
    Bdqid: 0x982dfe120000ae66\r\n
    Cache-Control: private\r\n
    Connection: keep-alive\r\n
    Content-Encoding: gzip\r\n
    Content-Type: text/html; charset=utf-8\r\n
    Cxy_all: baidu+d5ffd56ee15695c2945bfcaa1d1800a4\r\n
    Cxy_ex: 1603784503+2185852107+2a8fd0c78bb0946aa4b2c251e334c70c\r\n
    Date: Tue, 27 Oct 2020 07:41:43 GMT\r\n
    Is_status: 0\r\n
    Server: BWS/1.1\r\n

```

DNS

As it is hard for humans to remember IP address, DNS is used to translate domain names to IP address, such as `www.google.com` → `72.17.11.228`

DNS is always used behind the scenes when you browse the internet or play online games where your game client tries to access the game server

Similar to HTTP, DNS is also Client / Server based. A DNS client, usually your computer, sends a DNS query to a DNS server, asking "What is the IP address of the domain called `www.google.com`?" The DNS server responds with a DNS reply containing the IP address

DNS Query

Query

- FQDN
 - Fully qualified domain name
 - mymail.somecollege.edu (Host name **mymail** in the domain **somecollege.edu**)
- Type
 - A - translate from a domain name to an IPv4 address
- Class
 - IN - Internet Network

```
Queries
www.google.com: type A, class IN
Name: www.google.com
[Name Length: 14]
[Label Count: 3]
Type: A (Host Address) (1)
Class: IN (0x0001)
```

DNS Reply

The reply repeats the question and appends to it the relevant answer with the IP address

DNS Reply

- Domain Name
- Type
- Class
- Value
 - Length + Address

```
Answers
www.google.com: type A, class IN, addr 173.194.113.81
Name: www.google.com
Type: A (Host Address) (1)
Class: IN (0x0001)
Time to live: 232
Data length: 4
Address: 173.194.113.81
```

As mentioned before, HTTPS communication is encrypted and the request and response cannot be understood from Wireshark. However, you can see which sites someone is browsing by correlating DNS queries that were performed right before an HTTPS request.

Guided example

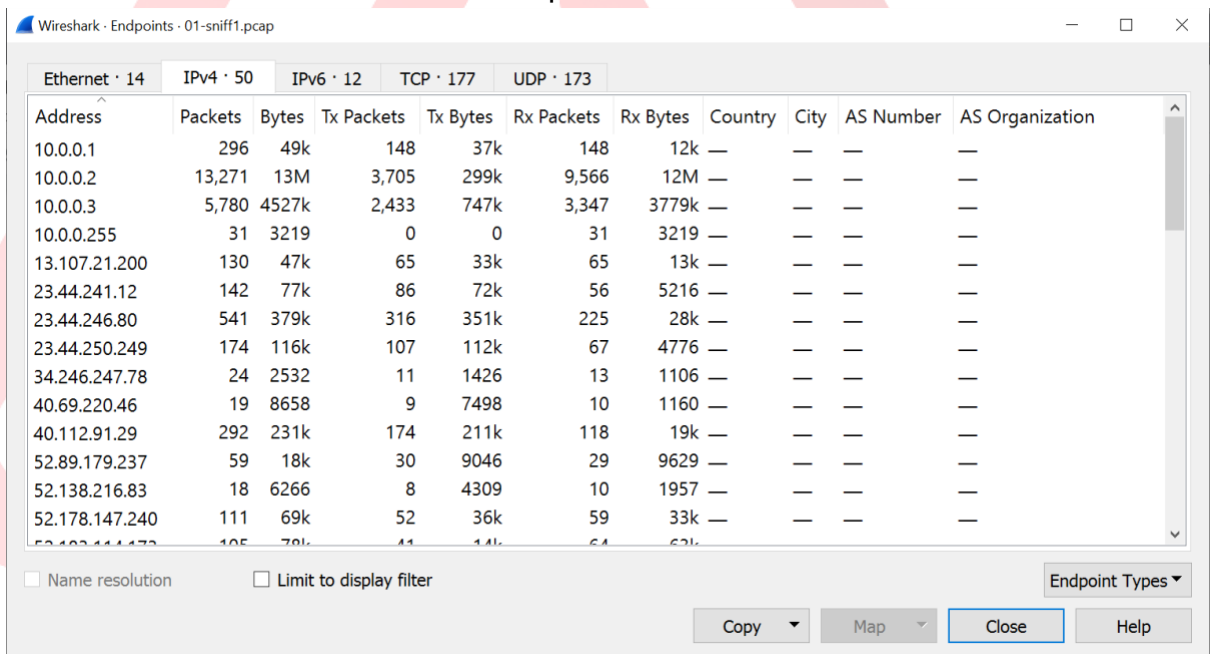
John came home one night and started browsing the web. Not doing much, suddenly his computer shuts down. Luckily, his son, Mike, had a sniffer running on John's computer. Investigate the result pcap file (sniff1.pcap) and answer the following questions.

Task

1. Describe John's network
 1. IP Addresses (in the LAN or WAN)
 2. MAC Addresses (in the LAN or WAN)
2. Describe John's computer:
 1. How do you know it is John's computer?
 2. Operating System
 3. Web browser

Let's load the given pcap file to Wireshark and provide the necessary information for this question using the existing analysis tools in Wireshark.

1. Using Wireshark Endpoints statistics we can easily get a list of all of the IP address and MAC addresses in the capture file.



Wireshark · Endpoints · 01-sniff1.pcap

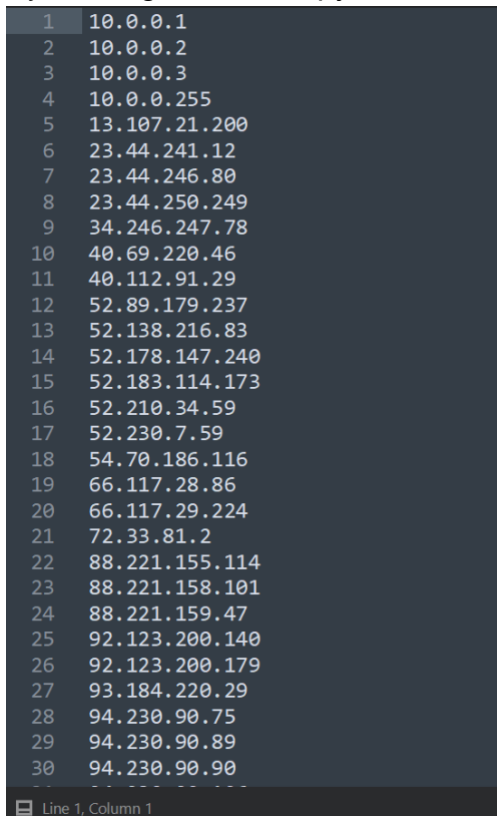
Ethernet · 14 IPv4 · 50 IPv6 · 12 TCP · 177 UDP · 173

Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	AS Number	AS Organization
10.0.0.1	296	49k	148	37k	148	12k	—	—	—	—
10.0.0.2	13,271	13M	3,705	299k	9,566	12M	—	—	—	—
10.0.0.3	5,780	4527k	2,433	747k	3,347	3779k	—	—	—	—
10.0.0.255	31	3219	0	0	31	3219	—	—	—	—
13.107.21.200	130	47k	65	33k	65	13k	—	—	—	—
23.44.241.12	142	77k	86	72k	56	5216	—	—	—	—
23.44.246.80	541	379k	316	351k	225	28k	—	—	—	—
23.44.250.249	174	116k	107	112k	67	4776	—	—	—	—
34.246.247.78	24	2532	11	1426	13	1106	—	—	—	—
40.69.220.46	19	8658	9	7498	10	1160	—	—	—	—
40.112.91.29	292	231k	174	211k	118	19k	—	—	—	—
52.89.179.237	59	18k	30	9046	29	9629	—	—	—	—
52.138.216.83	18	6266	8	4309	10	1957	—	—	—	—
52.178.147.240	111	69k	52	36k	59	33k	—	—	—	—

☐ Name resolution ☐ Limit to display filter Endpoint Types ▼

Copy Map Close Help

By clicking on the “Copy” button, we can copy the information in a CSV format



```
1 10.0.0.1
2 10.0.0.2
3 10.0.0.3
4 10.0.0.255
5 13.107.21.200
6 23.44.241.12
7 23.44.246.80
8 23.44.250.249
9 34.246.247.78
10 40.69.220.46
11 40.112.91.29
12 52.89.179.237
13 52.138.216.83
14 52.178.147.240
15 52.183.114.173
16 52.210.34.59
17 52.230.7.59
18 54.70.186.116
19 66.117.28.86
20 66.117.29.224
21 72.33.81.2
22 88.221.155.114
23 88.221.158.101
24 88.221.159.47
25 92.123.200.140
26 92.123.200.179
27 93.184.220.29
28 94.230.90.75
29 94.230.90.89
30 94.230.90.90
```

and in a text editor we can go over the list and analyze it.

We can see that some of the IPs are internal private IPs which are IPs in John’s LAN. Others are public Internet (WAN) IPs.

We can also see that some of the IPs are not real IPs like the following:
10.0.0.255 - A IP used for broadcasting a message to the entire Network Subnet
224.0.0.22 , 224.0.0.252 - Multicast IPs which are used to send data to a group of hosts in a computer network.

After removing these 3 IPs, we are left with a list of 47 IPs containing LAN and WAN IP addresses.

2. For the list of MAC addresses we can use again the Endpoints statistics but this time too look at the Ethernet protocol.

```

1  00:0c:29:31:f9:66
2  00:0c:29:4c:0f:cd
3  00:0c:29:b9:45:b2
4
5  01:00:5e:00:00:16
6  01:00:5e:00:00:fc
7
8  33:33:00:00:00:01
9  33:33:00:00:00:02
10 33:33:00:00:00:0c
11 33:33:00:00:00:16
12 33:33:00:01:00:02
13 33:33:00:01:00:03
14
15 33:33:ff:4c:0f:cd
16 33:33:ff:ef:1a:ce
17
18 ff:ff:ff:ff:ff:ff

```

We can now start to analyze the MAC addresses as we did with the IP addresses.

We first divide them to groups with the same first 3 bytes of the MAC address which represent the OUI (Organization Unique Identifier) of the network device. Then we can go over the list and try to see what MAC addresses represent real devices and what are special addresses for broadcasting or multicasting.

ff:ff:ff:ff:ff:ff - This MAC address is easily identified as broadcast address and can be eliminated from the list.

Now to the 33:33:ff group:

eth.addr == 33:33:ff:4c:0f:cd					
No.	Time	Source	Destination	Protocol	SPort
425	23:31:31.331567	::	ff02::1:ff4c:fcd	ICMPv6	
664	23:31:32.029629	::	ff02::1:ff4c:fcd	ICMPv6	

>	Frame 425: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
>	Ethernet II, Src: VMware_4c:0f:cd (00:0c:29:4c:0f:cd), Dst: IPv6mcast_ff:4c:0f:cd (33:33:ff:4c:0f:cd)
>	Internet Protocol Version 6, Src: ::, Dst: ff02::1:ff4c:fcd
>	Internet Control Message Protocol v6

Wireshark helps us by recognizing this OUI (33:33:ff) as a special MAC address used for multicast. This means it is not a real MAC address of a device in the network and all of the MAC addresses with this OUI can be eliminated.

Same thing happens with the 33:33:00 group:

eth.addr == 33:33:00:00:00:01					
No.	Time	Source	Destination	Protocol	
2295	23:31:45.737945	fe80::299f:e5c5:abef:1ace	ff02::1	ICMPv6	
<					
> Frame 2295: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)					
> Ethernet II, Src: VMware_4c:0f:cd (00:0c:29:4c:0f:cd), Dst: IPv6mcast_01 (33:33:00:00:00:01)					
> Internet Protocol Version 6, Src: fe80::299f:e5c5:abef:1ace, Dst: ff02::1					
> Internet Control Message Protocol v6					

And with 01:00:5e group:

eth.addr == 01:00:5e:00:00:16					
No.	Time	Source	Destination	Protocol	S
2314	23:31:50.350363	10.0.0.3	224.0.0.22	IGMPv3	
2316	23:31:50.456267	10.0.0.3	224.0.0.22	IGMPv3	
2318	23:31:50.480789	10.0.0.3	224.0.0.22	IGMPv3	
<					
> Frame 2314: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)					
> Ethernet II, Src: VMware_4c:0f:cd (00:0c:29:4c:0f:cd), Dst: IPv4mcast_16 (01:00:5e:00:00:16)					
> Internet Protocol Version 4, Src: 10.0.0.3, Dst: 224.0.0.22					
> Internet Group Management Protocol					

So all of these MACs can be eliminated as well.

Finally, we are left with the 00:0c:29 group. Looking at the traffic coming from or going to this MAC address we can see that MAC addresses in this group are addresses of actual devices in the network. Specifically in this case, Virtual devices created with VMware virtualization software.

eth.addr == 00:0c:29:b9:45:b2					
No.	Time	Source	Destination	Protocol	SP
2306	23:31:48.690789	10.0.0.2	104.25.198.31	TCP	4
2307	23:31:48.690981	104.25.198.31	10.0.0.2	TLSv1.3	
2308	23:31:48.690982	104.25.198.31	10.0.0.2	TLSv1.3	
<					
> Frame 2307: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)					
> Ethernet II, Src: VMware_31:f9:66 (00:0c:29:31:f9:66), Dst: VMware_b9:45:b2 (00:0c:29:b9:45:b2)					
> Internet Protocol Version 4, Src: 104.25.198.31, Dst: 10.0.0.2					
> Transmission Control Protocol, Src Port: 443, Dst Port: 49224, Seq: 712203, Ack: 4319, Len: 32					
> Transport Layer Security					

So to answer this part of the question, MAC address in John's network are:

00:0c:29:31:f9:66

00:0c:29:4c:0f:cd

00:0c:29:b9:45:b2

For part two of this exercise we need to focus on John's computer. For this part we will have to understand which computer is John's computer. We have seen 3 IP

addresses. We know one of them, 10.0.0.1, is the router's IP while John's computer can be either 10.0.0.2 or 10.0.0.3.

Let's try to find the information of the OS and Web browser.

Usually, a header called User-Agent that is being sent with every HTTP request can provide us with this information. So let's filter on HTTP protocol.

http					
No.	Time	Source	Destination	Protocol	SP
21	23:31:20.768264	10.0.0.2	94.230.90.89	HTTP	4
25	23:31:20.782070	94.230.90.89	10.0.0.2	HTTP	4
46	23:31:22.810904	10.0.0.2	93.184.220.29	OCSP	4
48	23:31:22.884598	93.184.220.29	10.0.0.2	OCSP	4

> Ethernet II, Src: VMware_b9:45:b2 (00:0c:29:b9:45:b2), Dst: VMware_31:f9:66 (00:0c:29:31:f9:66)

> Internet Protocol Version 4, Src: 10.0.0.2, Dst: 94.230.90.89

> Transmission Control Protocol, Src Port: 49210, Dst Port: 80, Seq: 1, Ack: 1, Len: 292

> Hypertext Transfer Protocol

> GET /success.txt HTTP/1.1\r\nHost: detectportal.firefox.com\r\nUser-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:61.0) Gecko/20100101 Firefox/61.0\r\nAccept: */*\r\n

We can take this User-Agent value to some websites that parse User-Agents for us:

Parse a User agent:

Mozilla/5.0 (Windows NT 6.1; WOW64; rv:61.0) Gecko/20100101 Firefox/61.0

Parse this user agent

[Parse your own user agent instead?](#)

Ads by Google

Here's how we parse the user agent:

Mozilla/5.0 (Windows NT 6.1; WOW64; rv:61.0) Gecko/20100101 Firefox/61.0



Firefox 61 on Windows 7

From this we learn that this is Firefox version 61 running on a Windows 7 machine. Let's see what information we can get about the other IP address (10.0.0.3).

ip.addr == 10.0.0.3 and http						
No.	Time	Source	Destination	Protocol	SPort	DPort
8782	23:32:38.631507	10.0.0.3	157.55.109.226	HTTP	49702	80
8787	23:32:38.829611	157.55.109.226	10.0.0.3	HTTP	80	49702
<div> <div> <div>></div> <div>Frame 8782: 1276 bytes on wire (10208 bits), 1276 bytes captured (10208 bits)</div> </div> <div> <div>></div> <div>Ethernet II, Src: VMware_4c:0f:cd (00:0c:29:4c:0f:cd), Dst: VMware_31:f9:66 (00:0c:29:31:f9:66)</div> </div> <div> <div>></div> <div>Internet Protocol Version 4, Src: 10.0.0.3, Dst: 157.55.109.226</div> </div> <div> <div>></div> <div>Transmission Control Protocol, Src Port: 49702, Dst Port: 80, Seq: 119, Ack: 1, Len: 1222</div> </div> <div> <div>></div> <div>[2 Reassembled TCP Segments (1340 bytes): #8781(118), #8782(1222)]</div> </div> <div> <div>></div> <div>Hypertext Transfer Protocol</div> <div> <div>></div> <div>POST /UploadData.aspx HTTP/1.1\r\n</div> <div> <div>Connection: Keep-Alive\r\n</div> <div>User-Agent: MSDW\r\n</div> <div>Content-Length: 1222\r\n</div> </div> </div> </div> </div>						

We can find different HTTP requests with different user agents such as: MSDW, MICROSOFT_DEVICE_METADATA_RETRIEVAL_CLIENT and Microsoft-CryptoAPI/10.0

These all seem not related to activities of a real user, but instead, HTTP requests that are being sent from some running Microsoft applications on this computer.

We can assume that 10.0.0.2, which contains user activity of someone browsing the Internet, is John's computer.

So to conclude:

Computer 00:0c:29:b9:45:b2 with IP 10.0.0.2 is John's computer. It is running a Windows 7 with Firefox web browser.

Exercise

Use the knowledge you've acquired in the tutorial and extract from the file red_alpha.pcap a PDF file which contains the password to complete this challenge