# **PROG20326 A3 Assignment**

## **Python Program to read and interpret a Hex file that contain “computer talk”**

**Python Concepts:** opening and reading a file, strings, lists, decisions, loops, conversion of data type and formatting and presenting an output.

The purpose of this assignment is to reinforce the content from the sections related to Python fundamentals such as reading and processing a text file, dealing with strings and lists, using while and for loops, making decision with conditionals and converting data formats.

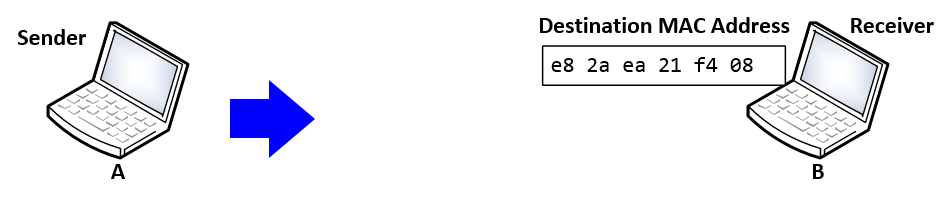
Several text files containing network traffic are provided. These text files contain ASCII Hex characters which represent actual information exchanged between computers. In the process, some network fundamentals concepts are also learned.

Computer traffic is encapsulated according to functional layers. Let’s see an example. This is a hex file that was sent by a computer. It is in hexadecimal and each separated word is one byte (8 binary bits).

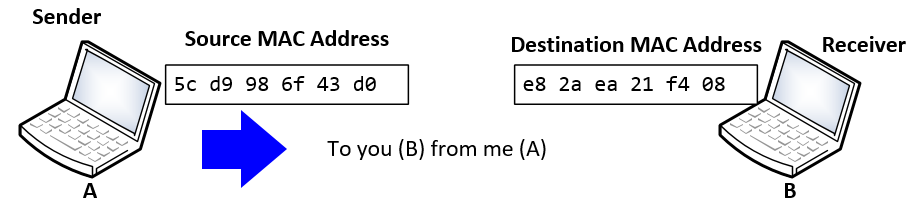
e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 08 00 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01 c0 a8 00 66 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

Computer sender converts the previous hex binary bits and then transmits them via a wire using electrical signals. (the same idea happens with wireless but the structure of the data would be different)

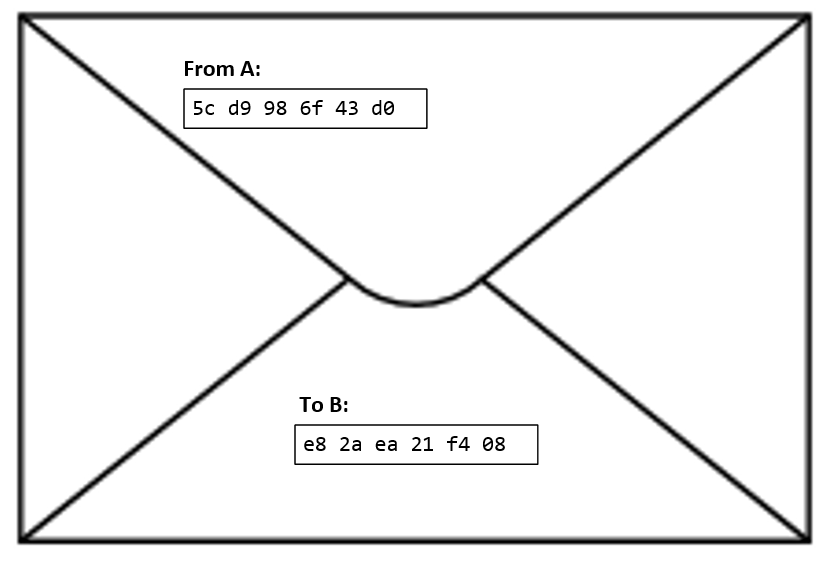
The first 6 bytes in the hex file **always** represents the **destination MAC address**. A MAC address is a Hex number of 6 bytes that uniquely identify a physical network interface. MAC addresses are unique and they are embedded on the interfaces.



The next 6 bytes **always** represents the sender’s (or source) MAC address.



So, this is like an envelope that contains destination and sender.

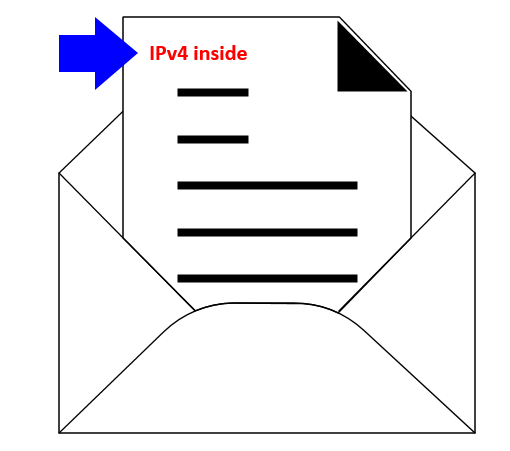


* This envelope is the **Frame** of the **Ethernet Protocol** (Layer 2)

Going back to the hex file, the first 12 bytes have been interpreted. The next two bytes have special meaning. This two-byte field is called the **Type** and it is a **pointer** to what is inside the envelope (the payload of the Ethernet protocol).

e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 **08 00** 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01 c0 a8 00 66 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

The computers (and any device that have network connectivity) read those two bytes, look them up in a table and depending of the value, they make a decision. **If** the value is **08 00**, as it is in this case, the next protocol inside the envelope is the **Internet Protocol** (IP version 4.)



The Internet Protocol has a header (just another envelope) of **20 bytes** by default. All of these bytes have a meaning, but in this assignment, we are only interested in a few of them.

The IP protocol header begins right after the **type** with the number 45 (4 in fact means version 4). If you count 20 bytes, that would be the IP header.

e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 **08 00 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01** **c0 a8 00 66** 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

Then, if you count from 45 (included) 10 bytes, read that value:

e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 **08 00 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01** **c0 a8 00 66** 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

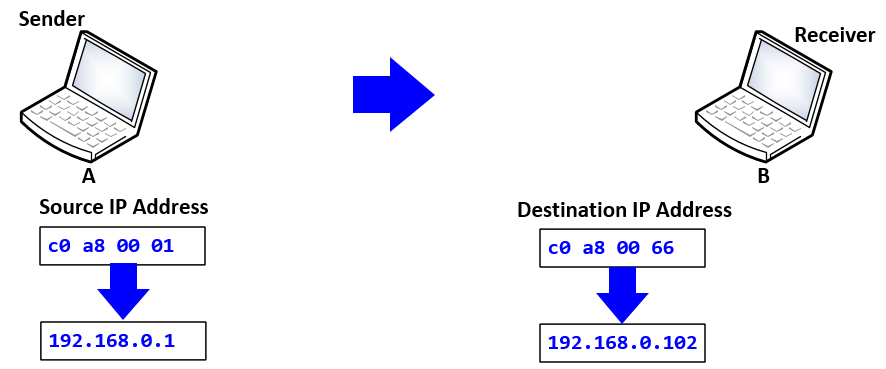
It is 06, assume that you are a computer, you have to think logically, what is **06**? This position **10** in the IP header is called the **Protocol Number** and it is a **Pointer** to the next encapsulated protocol. The computer reads the **06** and look it up in the next table. **If** the number is **06**, then the next encapsulated protocol is **TCP** and that is this case as the table shows:

|  |  |  |
| --- | --- | --- |
| **Protocol (Hex)** | **Protocol Number (in Decimal)** | **Encapsulated Protocol** |
| 1 | 1 | Internet Control Message ICMP |
| 2 | 2 | Internet Group Management IGMP |
| **6** | **6** | **Transmission Control Protocol TCP** |
| 11 | 17 | Transport User Datagram UDP |
| 58 | 88 | Routing Protocol EIGRP |
| 59 | 89 | Routing Protocol OSPF |

There are two more fields of interest for this assignment, the final 8 bytes of the IPv4 header contain: first, the source IP address (4 bytes) and then the destination IP address (4 bytes).

e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 **08 00 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01** **c0 a8 00 66** 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

So, c0 a8 00 01 is the IP address of the source and c0 a8 00 06 is the IP address of the destination. Those hex numbers converted to decimal are 192.168.0.1 and 192.168.0.102. (I sniff this data at home).



After the IP header comes the TCP protocol in this case. The TCP protocol also has a header that looks like this:

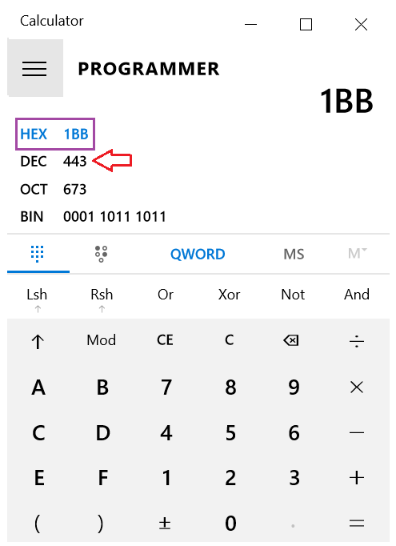
**TCP Header**



The first 2 bytes represent the **source port number** and the next two bytes represent the **destination port number**. These are pointers to the ports carrying the conversation. In our data:

e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 **08 00** 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01 c0 a8 00 66 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

The source port is 01 bb which converted to decimal is 443:



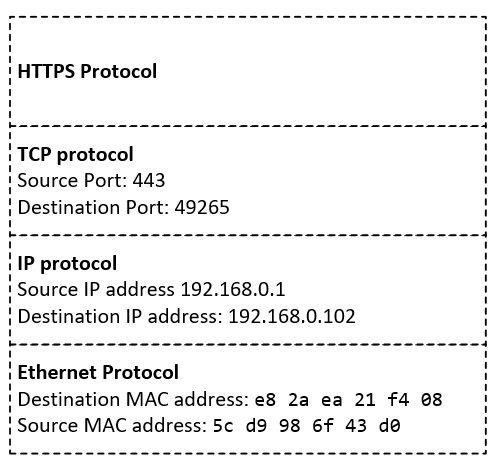
The computer looks the number up in a table:

**L4 (Port Numbers) for Encapsulated Protocols**

|  |  |  |
| --- | --- | --- |
| Port Number (in hex) | Port Number (in decimal) | Next Encapsulated Protocol |
| 14 | 20 and 21 | File Transfer Protocol FTP |
| 16 | 22 | Secure Shell SSH |
| 17 | 23 | TELNET |
| 19 | 25 | Email SMTP |
| 35 | 53 | Domain Name System DNS |
| 50 | 80 | Hypertext Transfer Protocol HTTP |
| 8F | 143 | Internet Message Access Protocol IMAP |
| A1 | 161 | Simple Network Management Protocol SNMP |
| B3 | 179 | Routing Border Gateway Protocol BGP |
| 02 08 | 520 | Routing Information Protocol RIP |
| **01 BB** | **443** | **Secure HTTP (HTTPS)** |
| 03 87 | 903 | VMware Remote Console |

Finally, the encapsulated protocol is a ciphered (secure) HTTPS which is for web traffic.

In summary, from the lower layer (Ethernet) going up to the application HTTPS:



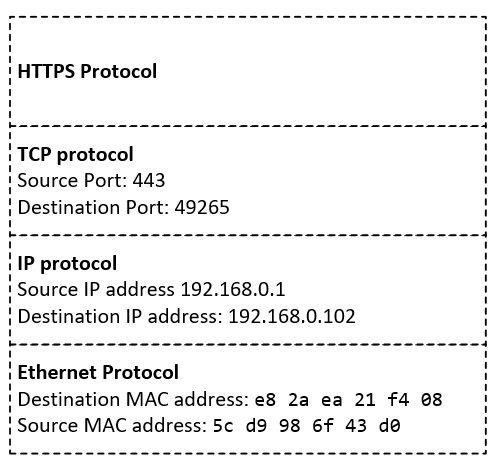
**The assignment:**

Write a Python code that:

* **Reads text files** that contain Internet traffic.
* **Process the file** to read the significant fields as explained before.
* Process and **interpret** the fields.
* **Print out** the encapsulation.
* Basically, if I give you this:

e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 08 00 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01 c0 a8 00 66 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

* You give me a program that does this (printout includes data formatting or presentation):



* Note: you might be creative in the format, in fact, that is a good thing.

These are the three files that your Python program have to read and interpret:

e8 2a ea 21 f4 08 5c d9 98 6f 43 d0 08 00 45 00 00 34 60 5f 00 00 3d 06 32 be c0 a8 00 01 c0 a8 00 66 01 bb c0 71 6a 98 3a 81 d3 9d 80 63 80 12 72 10 57 02 00 00 02 04 05 b4 01 01 04 02 01 03 03 07

ca 00 08 0c 00 1d ca 01 08 0c 00 1d 08 00 45 00 00 3f 03 33 00 00 7f 11 b4 26 c0 a8 02 03 c0 a8 01 01 f1 14 00 35 00 2b 37 ad a7 87 01 00 00 01 00 00 00 00 00 00 03 63 72 6c 09 6d 69 63 72 6f 73 6f 66 74 03 63 6f 6d 00 00 01 00 01

3c 97 0e 1a 23 9f 00 08 e3 ff fd a0 08 00 45 00 00 6a 88 b7 00 00 3e 01 07 a8 8e 37 e9 1a 8e 37 e6 aa 03 03 e9 7c 00 00 00 00 45 00 00 4e 2d fa 00 00 7e 11 22 71 8e 37 e6 aa 8e 37 e9 1a 00 89 00 89 00 3a ea 41 e6 83 01 10 00 01 00 00 00 00 00 00 20 46 48 46 41 45 42 45 45 43 41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 41 41 00 00 20 00 01

In the appendix, you have the tables and headers with more information that is needed to interpret the data.

**Tips and Tricks:**

* For this lab you can copy paste wireshark hexa output for a single packet onto a text file called computer\_talk
* Prepare dictionaries or lists to implement the tables given in the appendix
* You can use seek(method) to the required position in the file and then run a loop to access subsequent bytes
* After finding the byte value, you can use them as indices of lists or convert them to string and use them as keys to lookup dictionaries.
* You will need the ASCII tables for DNS queries and responses.

**Competencies:**

* Use of data structures (lists, dictionaries)
* Writing to and Reading from a File
* Use of selection and iteration statements

**Submit to SLATE:**

* The Python program with your name in the documentation.
* A PDF with the output of the testing of the three cases.

**Assignment Appendix**

**Protocol Headers**

**Rules:**

* One Byte is represented by 8 bits.
* Start de-encapsulating from the first byte in the data file. That corresponds to the first byte of the layer 2 protocol.
* IP header version field is always 4.

**Ethernet Header**

Destination MAC Address Source MAC Address Encapsulated Type

**IP Version 4 Header**



**UDP Header**



**TCP Header**



**Ethernet Types**

**(Protocols Encapsulated by Ethernet)**

“The 13th and 14th octets of an Ethernet or IEEE 802.3 packet (after the preamble) consist of the "Ethernet Type" or "IEEE802.3 Length" field.

|  |  |
| --- | --- |
| Type (in Hexadecimal) | Next Encapsulated Protocol |
| **08 00** | **Internet (IPv4)** |
| 86 DD | Internet (IPv6) |
| 08 06 | Address Resolution (ARP) |

**Some Protocols Encapsulated by IPv4**

**(Protocol Field of the IPv4 Header)**

|  |  |  |
| --- | --- | --- |
| **Protocol (Hex)** | **Protocol Number (in Decimal)** | **Encapsulated Protocol** |
| 1 | 1 | Internet Control Message ICMP |
| 2 | 2 | Internet Group Management IGMP |
| 6 | 6 | Transport Control Protocol TCP |
| 11 | 17 | Transport User Datagram UDP |

**Some Port Numbers as encapsulated by either TCP or UDP**

|  |  |  |
| --- | --- | --- |
| Port Number (in hex) | Port Number (in decimal) | Next Encapsulated Protocol |
| 14 | 20 and 21 | File Transfer Protocol FTP |
| 16 | 22 | Secure Shell SSH |
| 17 | 23 | TELNET |
| 19 | 25 | Email SMTP |
| 35 | 53 | Domain Name System DNS |
| 50 | 80 | Hypertext Transfer Protocol HTTP |
| 8F | 143 | Internet Message Access Protocol IMAP |
| A1 | 161 | Simple Network Management Protocol SNMP |
| B3 | 179 | Routing Border Gateway Protocol BGP |
| 02 08 | 520 | Routing Information Protocol RIP |
| 01 BB | 443 | Secure HTTP (HTTPS) |
| 03 87 | 903 | VMware Remote Console |

**Appendix**

**ASCII Table with most useful Hex characters**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Lower case** | | | **Upper Case** | | |
| **Hex** | **Meaning** |  | | **Hex** | **Meaning** |
| 61 | a |  | | 41 | A |
| 62 | b |  | | 42 | B |
| 63 | c |  | | 43 | C |
| 64 | d |  | | 44 | D |
| 65 | e |  | | 45 | E |
| 66 | f |  | | 46 | F |
| 67 | g |  | | 47 | G |
| 68 | h |  | | 48 | H |
| 69 | i |  | | 49 | I |
| 6A | j |  | | 4A | J |
| 6B | k |  | | 4B | K |
| 6C | l |  | | 4C | L |
| 6D | m |  | | 4D | M |
| 6E | n |  | | 4E | N |
| 6F | o |  | | 4F | O |
| 70 | p |  | | 50 | P |
| 71 | q |  | | 51 | Q |
| 72 | r |  | | 52 | R |
| 73 | s |  | | 53 | S |
| 74 | t |  | | 54 | T |
| 75 | u |  | | 55 | U |
| 76 | v |  | | 56 | V |
| 77 | w |  | | 57 | W |
| 78 | x |  | | 58 | X |
| 79 | y |  | | 59 | Y |
| 7A | z |  | | 5A | Z |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hex** | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| **Meaning** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

|  |  |  |
| --- | --- | --- |
| **Hex** | 20 | 2C |
| **Meaning** | space | comma |