Prodigy InfoTech

Packet Sniffing

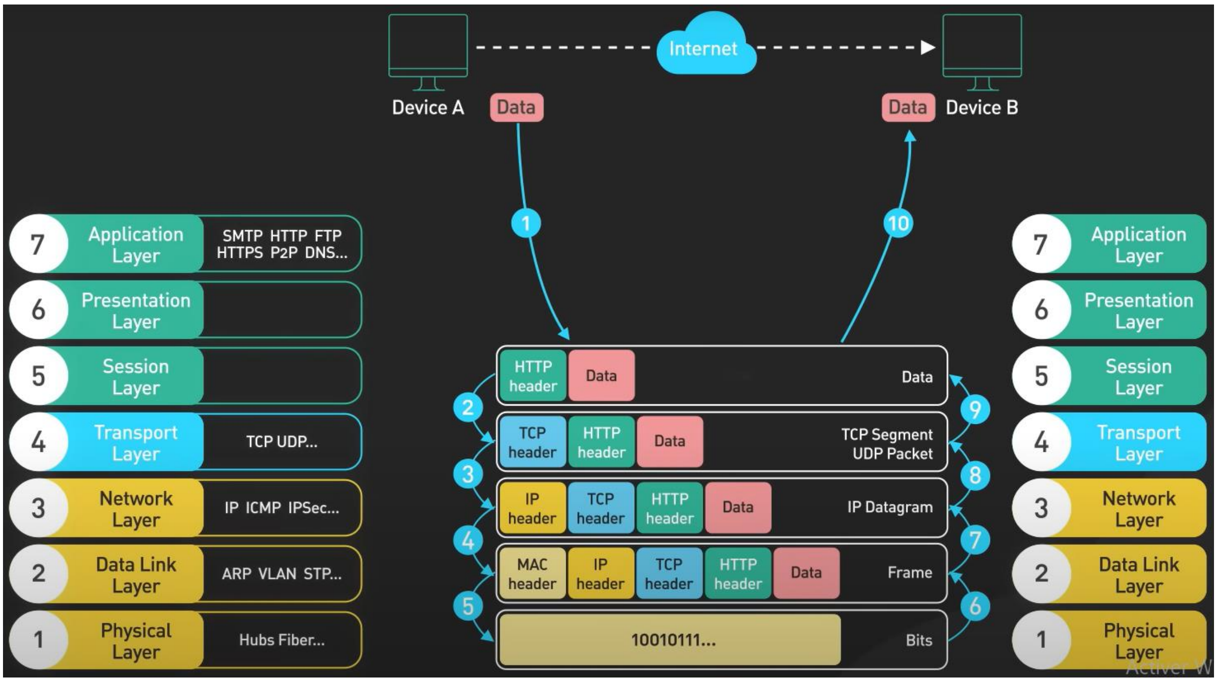
Developed By Soulaima Jaidane

Introduction to Layers of the OSI Model

1) **Physical layer :** Transmitting raw bits of data.
2) **Data Link layer :** Takes the raw data from the physical layer and organizes them into frames.
3) **Network layer :** Routing data frames across different networks (the IP of TCP/IP, ICMP).
4) **Transport layer :** Handles end-to-end communications between two nodes (TCP, UDP).

- **TCP :** Divides data into small segments and sends each segment individually. Each segment has a sequence number attached to it to reassemble the data in the correct order. Includes a checksum to ensure data integrity during transmission.
- **UDP :** Sends packets and the receiving end is responsible for determining whether the packets were received correctly. If errors occur, the receiver's end notifies the sender.

5,6 and 7) **Application** : HTTP handles the communication and data exchange between web browsers and servers.



Introduction to Packet Sniffers

what a packet sniffer is and why it is useful?

A packet sniffer, or network sniffer, allows us to monitor and analyze network traffic by capturing packets of data traveling over a network. This can be useful for network management, troubleshooting, and security(Attacks like DDOS cause lots of damage to the organisation Interrupting their workflow)

What is Ethernet Frame ? :

Ethernet Frame

When transmitting data such as an image or a web page over the internet, it is broken down into smaller units called frames. If an error occurs and a frame is not successfully transmitted, only the damaged frame needs to be resent, not the entire document. This efficiency is a key advantage of using frames.

Ethernet is the most widely used network technology and its frame structure consists of several key elements:

1. **Preamble**: Synchronizes the receiver's NIC (Network Interface Card) to ensure it can read the in coming data correctly.always ending with 11
2. **Destination MAC Address and Source MAC Address**:

   Data Link Layer (Layer 2). Identifies the intended recipient of the frame. One MAC address will be your computer, and the other will be the sender(router).

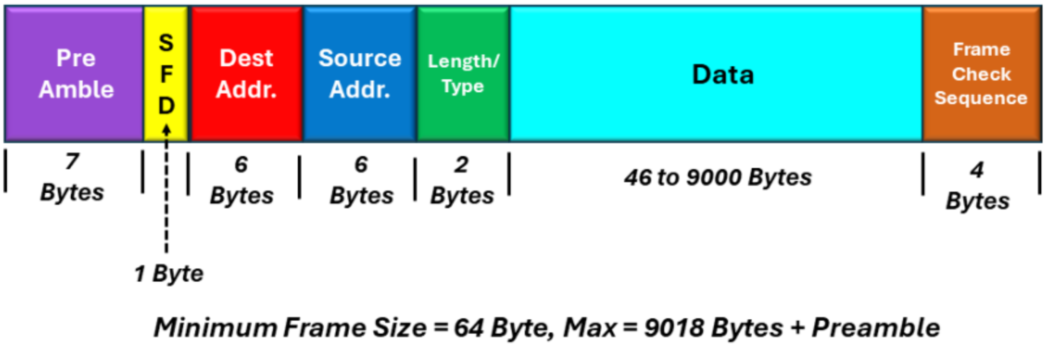
3. **EtherType/Length**: Indicates the type of **protocol** the data uses, such as IPv4 or IPv6.

   Helps the receiving device know how to process the encapsulated data..

4. **Data and Padding**: The actual data being transmitted. If the data is less than 46 bytes, padding is added to meet the minimum frame size requirement of 64 bytes.

5. **Frame Check Sequence (FCS)**

Uses a cyclic redundancy check (CRC) algorithm et verify the integrity of the data. If the FCS calculated by the receiver matches the sender's FCS, the frame is considered intact ; otherwise, it is discarded.



Sniffer_demo.py :

Part 1 : Formatting MAC address

The function get_mac_addr format the MAC adresse into a human readable format(AA:BB:CC:DD:EE:FF)

Parte 2 : Capturing traffic

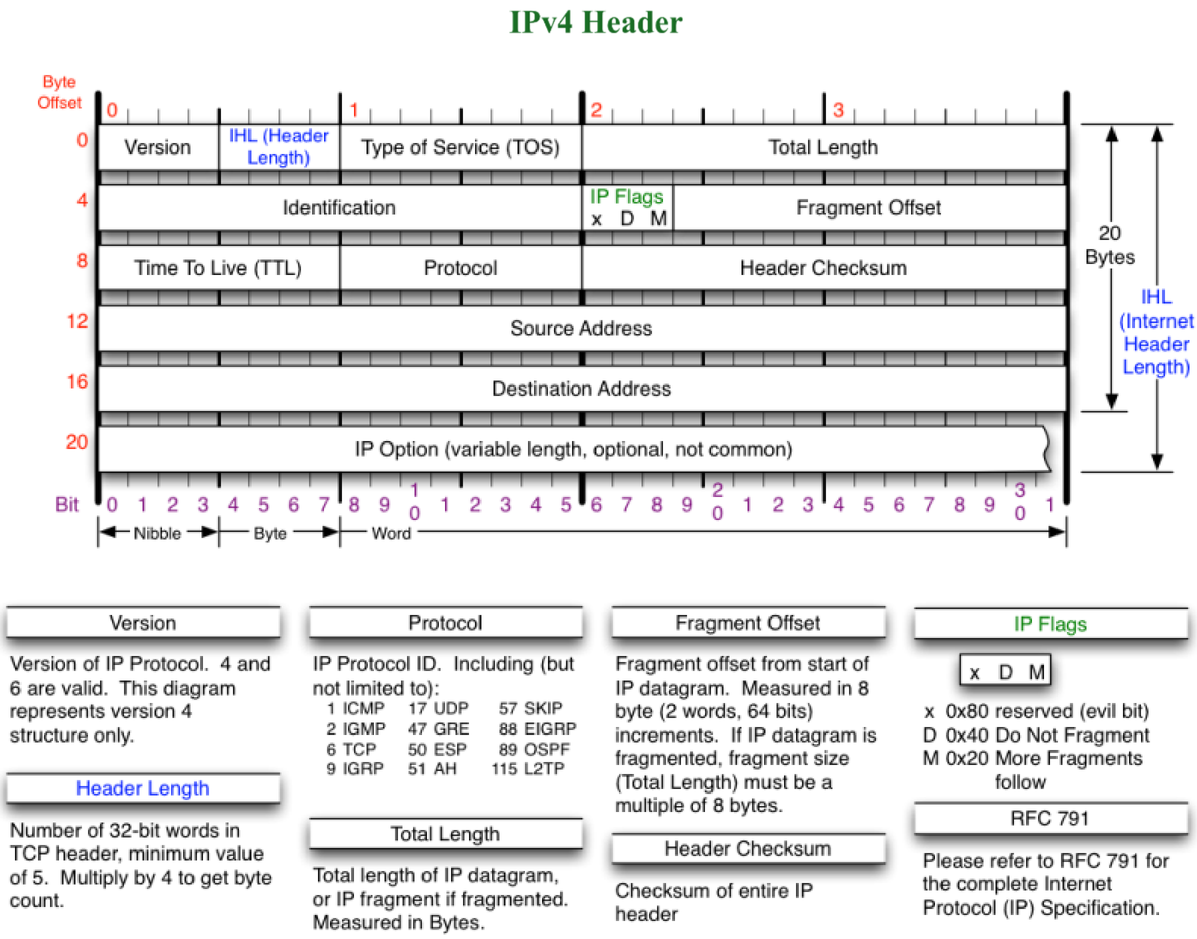
The main function is a loop listen to packets to come in and unpack the informaton from it and display it to the user

We used socket to have connections with other computers and have raw socket

- **Socket Configuration:**
  - **socket.AF_PACKET:** Works at the Ethernet packet level.
  - **socket.SOCK_RAW:** Accesses raw packets directly.
  - **socket.ntohs(3):** Ensures compatibility across different machines.

And when he crecieve a dataconn.recvform(655353) try to figure out addr MAC , Ethernet or IP protocol, and the actual data payload from each packet

Part 3:Unpacking IP Packet Headers



Version

Version of IP Protocol. 4 and 6 are valid. This diagram represents version 4 structure only.

Header Length

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

Protocol

IP Protocol ID. Including (but not limited to):

1 ICMP	17 UDP	57 SKIP
2 IGMP	47 GRE	88 EIGRP
6 TCP	50 ESP	89 OSPF
9 IGRP	51 AH	115 L2TP

Total Length

Total length of IP datagram, or IP fragment if fragmented. Measured in Bytes.

Fragment Offset

Fragment offset from start of IP datagram. Measured in 8 byte (2 words, 64 bits) increments. If IP datagram is fragmented, fragment size (Total Length) must be a multiple of 8 bytes.

Header Checksum

Checksum of entire IP header

IP Flags

x D M

x 0x80 reserved (evil bit)
D 0x40 Do Not Fragment
M 0x20 More Fragments follow

RFC 791

Please refer to RFC 791 for the complete Internet Protocol (IP) Specification.

What is an IP Header?

An IP header is a crucial part of the data packet that allows devices to connect to routers and the internet. It precedes the IP payload or data (the first 20 bytes of the packet) and contains important information about the packet.

We create a function ipv4_packet to extract the IP header and the data from an IP packet.

- **Key Components Extracted from the ip header:**
  - version: The version of the IP protocol.
  - header_length: The length of the header, used to find where the data starts.
  - ttl: Time to live, which indicates how long the packet is valid.
  - proto: The protocol used (e.g., TCP, UDP,ICMP).
  - ipv4(src): The source IP address.
  - ipv4(target): The destination IP address.

And we extracted Data: The actual data or payload, extracted as data[header_length:].

Part 4 :Unpacking ICMP ,TCP and UDP

In this part, we will identify the protocol type from the unpacked IPv4 header and use it to determine the type of data. We'll handle ICMP, TCP, and UDP protocols by creating separate methods for each and using conditional statements to call the appropriate unpacking method.

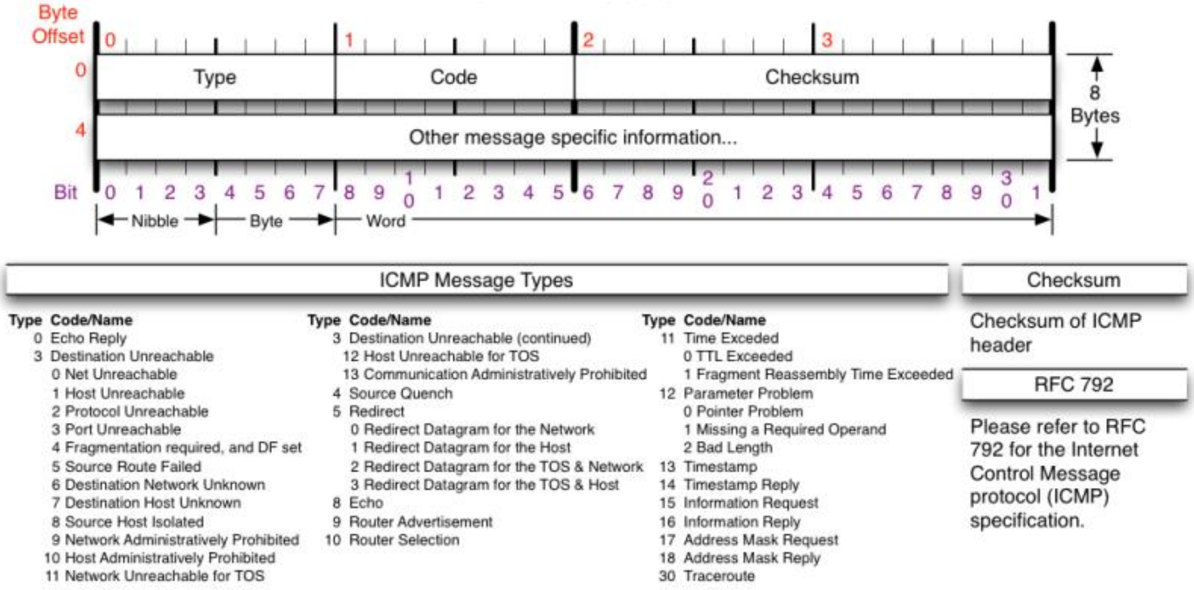
For example, if the protocol is 1 (ICMP), 6 (TCP), or 17 (UDP), we will call the respective unpacking methods

ICMP :

We used the function icmp_packet to unpack ICMP headers and data, converting them to decimal.

- **ICMP Header:** Contains type (1 byte: B), code (1 byte: B), and checksum (2 bytes: H).
- **Data:** The data starts after the header ends data[:4]

ICMP



Type	Code/Name
0	Echo Reply
3	Destination Unreachable
	0 Net Unreachable
	1 Host Unreachable
	2 Protocol Unreachable
	3 Port Unreachable
	4 Fragmentation required, and DF set
	5 Source Route Failed
	6 Destination Network Unknown
	7 Destination Host Unknown
	8 Source Host Isolated
	9 Network Administratively Prohibited
	10 Host Administratively Prohibited
	11 Network Unreachable for TOS

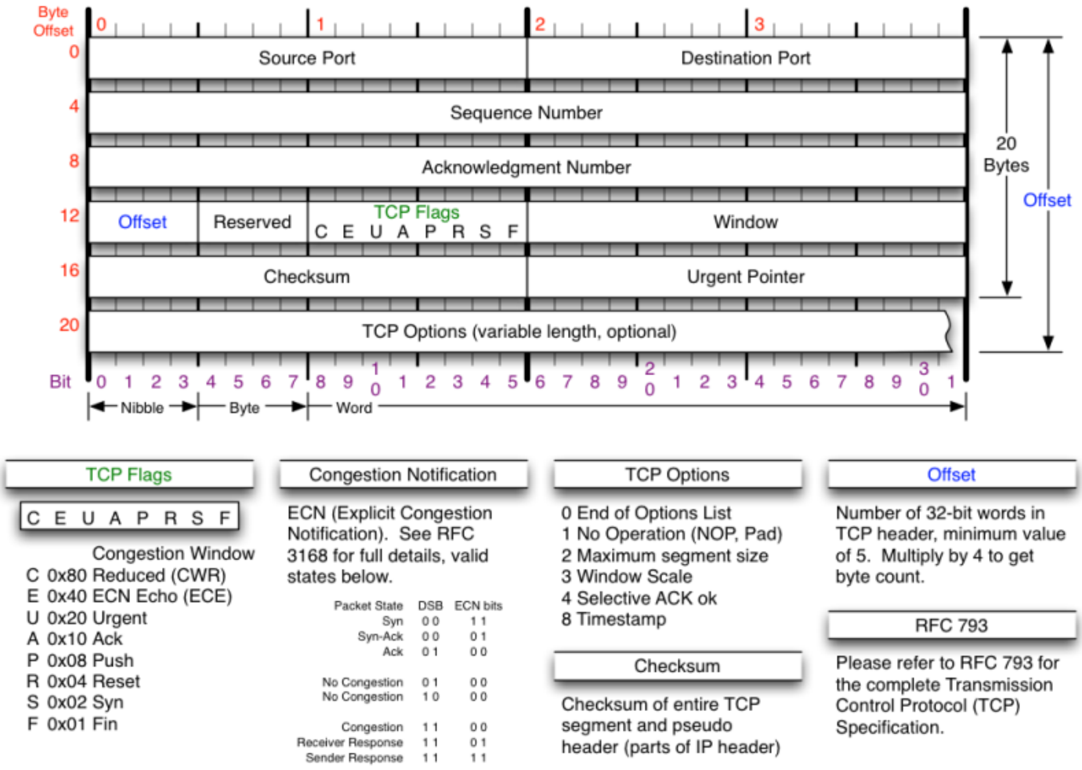
Type	Code/Name
3	Destination Unreachable (continued)
	12 Host Unreachable for TOS
	13 Communication Administratively Prohibited
4	Source Quench
5	Redirect
	0 Redirect Datagram for the Network
	1 Redirect Datagram for the Host
	2 Redirect Datagram for the TOS & Network
	3 Redirect Datagram for the TOS & Host
8	Echo
9	Router Advertisement
10	Router Selection

Type	Code/Name
11	Time Exceded
	0 TTL Exceeded
	1 Fragment Reassembly Time Exceeded
12	Parameter Problem
	0 Pointer Problem
	1 Missing a Required Operand
	2 Bad Length
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
30	Traceroute

Checksum of ICMP header

Please refer to RFC 792 for the Internet Control Message protocol (ICMP) specification.

TCP header



TCP Flags

C E U A P R S F

Congestion Window
C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Receiver Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

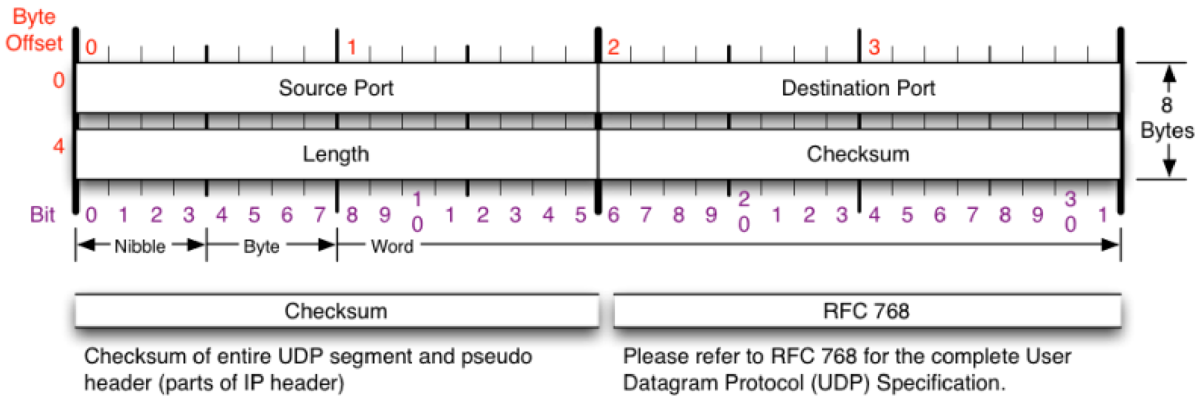
RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

TCP: We used the function tcp_segment to unpack TCP headers and data.

- **TCP Header:** Contains source port, destination port, sequence number, acknowledgment number, offset, reserved bits, and TCP flags.
  - **Offset Calculation :** We extract the offset from the reserved_offset field, which is 16 bits long. The offset itself occupies 4 bits within this field. By adding 12 (which accounts for the offset, reserved bits, and TCP flags) and multiplying by 4, we determine the offset size in bytes.
  - **Flags:** URG, ACK, PSH, RST, SYN, FIN.the uses three-way handhshake for establishing a TCP connection
- **Data:** The data starts after the calculated offset.

UDP



Checksum

Checksum of entire UDP segment and pseudo header (parts of IP header)

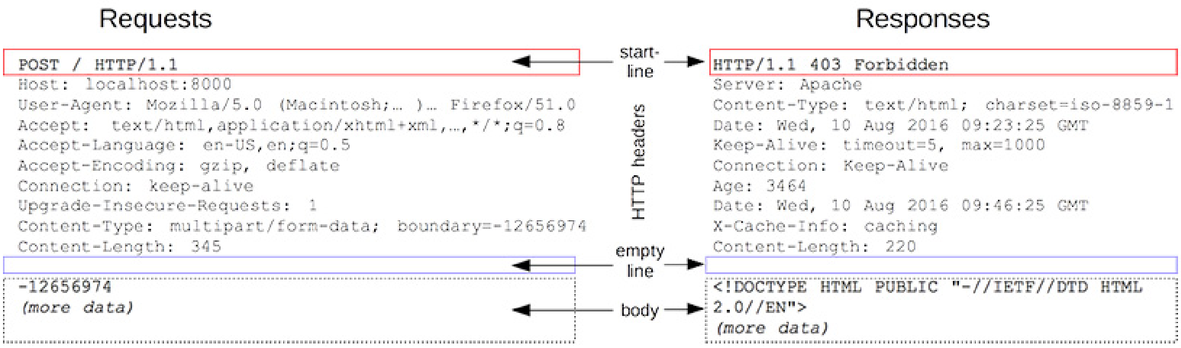
RFC 768

Please refer to RFC 768 for the complete User Datagram Protocol (UDP) Specification.

We used the function udp_segment to unpack UDP headers and data.

- **UDP Header:** Contains source port (2 bytes: H), destination port (2 bytes: H), and length (2 bytes: H).
- **Data:** The data starts after the header ends data[8:]
-

HTTP



```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

```
HTTP/1.1 403 Forbidden
Server: Apache
Content-Type: text/html; charset=iso-8859-1
Date: Wed, 10 Aug 2016 09:23:25 GMT
Keep-Alive: timeout=5, max=1000
Connection: Keep-Alive
Age: 3464
Date: Wed, 10 Aug 2016 09:46:25 GMT
X-Cache-Info: caching
Content-Length: 220

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
2.0//EN">
(more data)
```

We divided the HTTP packet , encapsulated in TCP, into four parts: first, we extract the HTTP method and URL. For POST requests, we additionally extract the data passed, after the empty line ,splitting it into form items like username and password, which are then displayed alongside the method and URL.

Part 5 : Displaying Packet Data

Displaying Packet Data for IPv4 when protocol(proto_eth)=8 (IPv4):

- IPv4 Data followed by TCP, UDP, or ICMP Data whent proto=1,6 or 17

Interface.py :

The interface.py script displays IPv4 packets categorized into TCP, UDP, ICMP, and HTTP protocols. HTTP packets show additional details such as user credentials. Ethernet packet types are also displayed. The interface allows searching by packet type, source IP address, and destination IP address. Upon completion, it creates an ipFile.txt file containing all destination IP addresses. It then invokes the locateIps.py script to generate the locations.kml file, mapping the geolocations of these IP addresses.

locateIPs.py :

This Python script uses APIs from IPHub and GeoPlugin to gather and process geographic and VPN-related data for a list of IP addresses stored in ipFile.txt.

Input Validation:

- Checks for the existence of ipFile.txt, which should contain IP addresses to process.

API Integration:

- Utilizes the IPHub API (http://v2.api.iphub.info/ip/) to determine if IP addresses are associated with VPN usage, providing ISP and VPN block status.

GeoPlugin API:

- Queries the GeoPlugin API (http://www.geoplugin.net/json.gp) to retrieve geographic details such as latitude, longitude, city, region, and country for each IP address.

Data Handling:

- Stores the retrieved data into arGeoResults list, ensuring robust handling of JSON decoding errors.

Output:

- Writes the collected geographic data into two CSV files:
  - geo-coords.csv for geographic coordinates.
  - geo-stats.csv for detailed geographic and VPN status information.

Visualization:

- Generates a KML file (locations.kml) using the SimpleKML library.
- Plots points for each valid location with detailed descriptions.
- Optionally connects them with a line if coordinates are available.

Start_sniffing.py :

This script uses pyfiglet with the bloody.fpf font style to display a bloody-themed banner for "Packet Sniffing" before prompting the user to choose an option to start the application.

Detecting the DDOS attack

Distributed denial-of-service DOS is a malicious attempt to disrupt the normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. For detectiing this type of major attack we are taking 3 factors into consideration that can confirm us that a DDOS has been performed:

- LOIC Download by the attacker
- Hivemind issued by the attacker
- No of packets sent by the attacker

Here we get into the TCP section of the IP and gets the data to see the HTTP header for confiramtion of the LOIC download and then we look for port 6667 i.e used by IRCservers for performing DDOS and get the command and next we get the count of the packets. This is done by utilizing the dpkt and socket library