Wireshark IP Packet Analysis: FastAPI File Upload

Aayush Adhikari, Roshan Tiwari, Shishir Sharma Rijal, Sudip Acharya

July 14, 2024

Introduction

This report analyzes an IP packet captured using Wireshark, focusing on the structure and content of the IP header. The capture was performed during a file upload to a FastAPI application on localhost.

1 FastAPI File Upload

FastAPI File Upload Implementation

To implement file upload in FastAPI, the following steps were taken:

```python
from fastapi import FastAPI, UploadFile, File, HTTPException, status
import os
import uuid

app = FastAPI()

UPLOAD_DIRECTORY = "uploads"
MAX_FILE_SIZE = 50 * 1024 * 1024  # 50MB

os.makedirs(UPLOAD_DIRECTORY, exist_ok=True)

@app.get("/")
def home():
    return {"message": "Welcome to FastAPI File Upload Example"}

@app.post("/upload/", status_code=status.HTTP_201_CREATED)
async def upload_file(file: UploadFile = File(...)):
    try:
        file_content = await file.read()

        file_size = len(file_content)
        if file_size > MAX_FILE_SIZE:
            raise HTTPException(
                status_code=status.HTTP_413_REQUEST_ENTITY_TOO_LARGE,
                detail=f"File size exceeds {MAX_FILE_SIZE // (1024 * 1024)}MB
                    limit",
            )


        unique_id = str(uuid.uuid4())
        file_extension = os.path.splitext(file.filename)[-1].lower()
        uploaded_file_name = f"{unique_id}{file_extension}"
        file_path = os.path.join(UPLOAD_DIRECTORY, uploaded_file_name)

        with open(file_path, "wb") as buffer:
            buffer.write(file_content)

        return {"file_name": uploaded_file_name, "file_size": file_size}

    except HTTPException as e:
        raise e
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Failed to upload file. Error: {e}",
        )

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="localhost", port=8000)
```

This code defines a FastAPI application with a file upload endpoint. The form in the HTML allows users to select and upload a file to the server.

2 Packet Capture Details

Capturing the Packet with Wireshark

The packet capture was performed using Wireshark with the following steps:

1. Start Wireshark and select the network interface used for localhost communication (e.g., ”Loopback: lo0” on Linux).
2. Begin the packet capture.
3. Perform the file upload to the FastAPI application by accessing the HTML form and submitting a file.
4. Stop the packet capture once the upload is complete.
5. Filter the captured packets to isolate the relevant IP packet(s) using a display filter such as ‘ip.src == 127.0.0.1 ip.dst == 127.0.0.1‘.

The captured packet details are analyzed in the following sections.

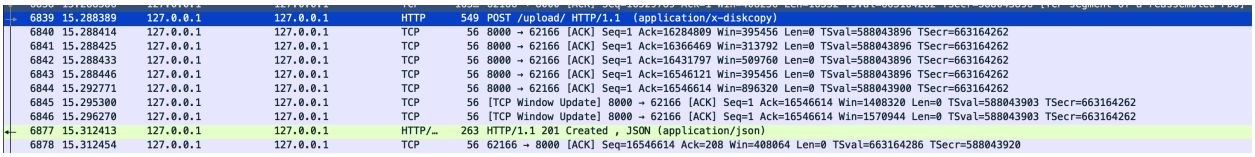

Figure 1: Packet file-upload

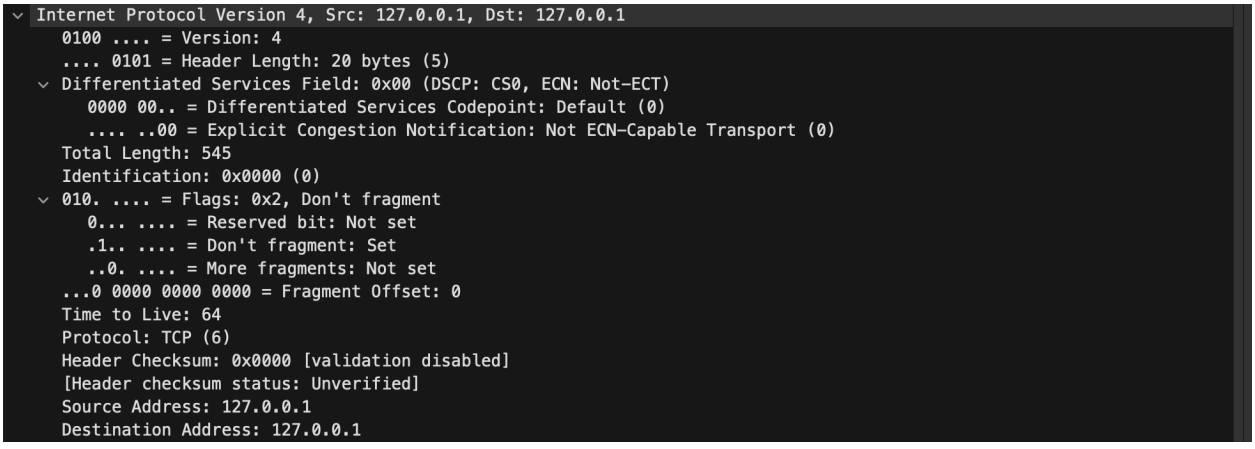

Figure 2: Header Fields

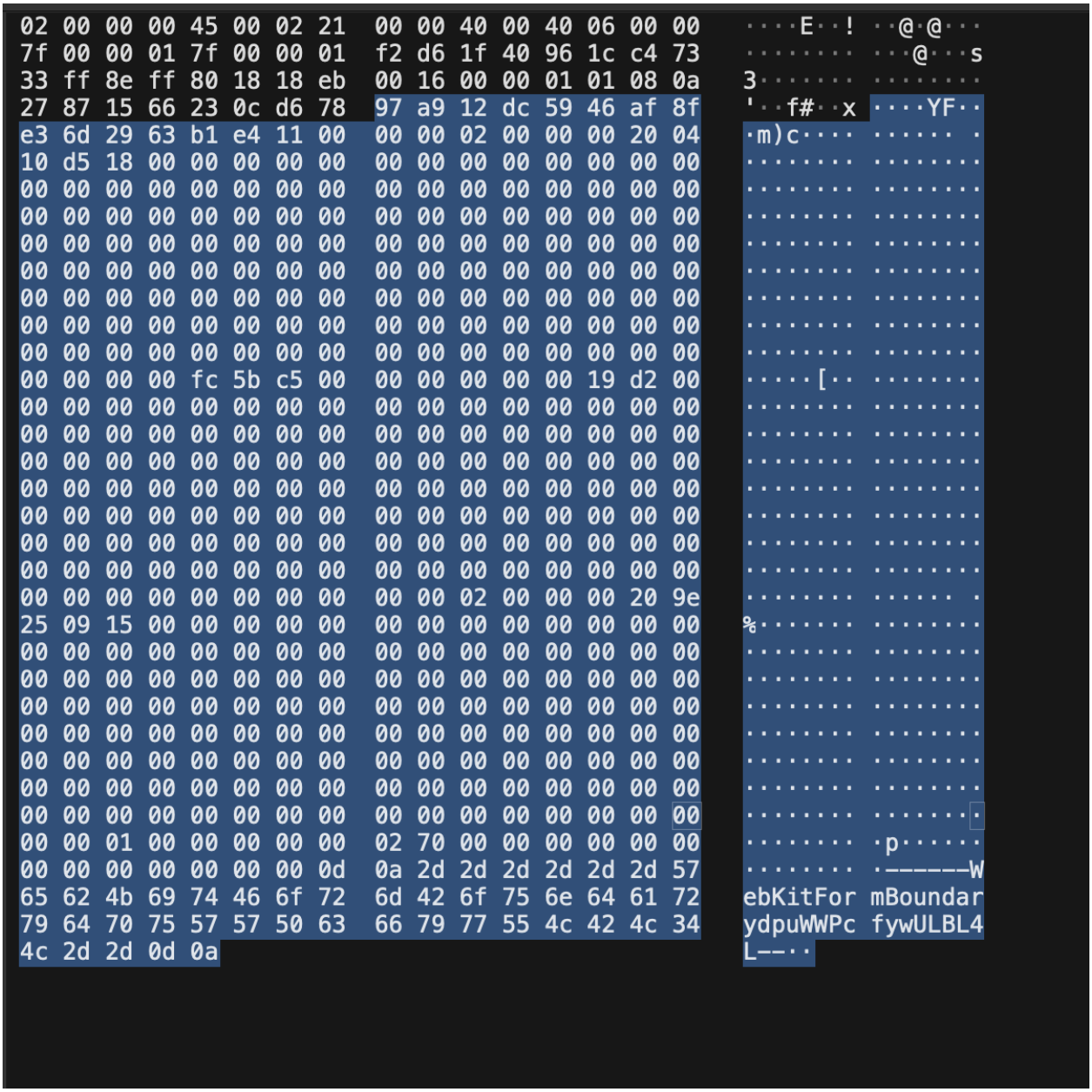

Figure 3: Hex

3 IP Header Data

IP Header (20 bytes)

```
45 00 02 21 00 00 40 00 40 06 00 00 7f 00 00 01 7f 00 00 01
```

4 IP Header Analysis

Field	Value (Hex)	Value (Decoded)	Explanation
Version	4	4	IPv4
IHL	5	5	Header length 20 bytes
DSCP	00	0	Default (CS0)
ECN	0	0	Not ECN-Capable Transport
Total Length	02 21	545 bytes	Packet size
Identification	00 00	0	Packet identifier
Flags	010	Don't Fragment	DF bit set
Fragment Offset	0000	0	No fragmentation
TTL	40	64	Max hops before discard
Protocol	06	6 (TCP)	Next level protocol
Header Checksum	00 00	0	Checksum (invalid/disabled)
Source IP	7f 00 00 01	127.0.0.1	Localhost
Destination IP	7f 00 00 01	127.0.0.1	Localhost

Table 1: IP Header Fields

5 Detailed Field Explanations

Field Descriptions

- **Version (4 bits)**: 4 indicates IPv4.
- **IHL (4 bits)**: 5 * 4 = 20 bytes, standard IPv4 header length.
- **DSCP (6 bits)**: 0 indicates default traffic class.
- **ECN (2 bits)**: 0 means ECN is not being used.
- **Total Length (16 bits)**: 545 bytes for the entire IP packet.
- **Identification (16 bits)**: 0, unused in this non-fragmented packet.
- **Flags (3 bits)**: 010 - Don't Fragment bit is set.
- **Fragment Offset (13 bits)**: 0, as the packet is not fragmented.
- **TTL (8 bits)**: 64, typical initial value for many systems.
- **Protocol (8 bits)**: 6 indicates TCP as the next layer protocol.
- **Header Checksum (16 bits)**: 0, indicating checksum is invalid or disabled for loopback.
- **Source IP (32 bits)**: 127.0.0.1, localhost address.
- **Destination IP (32 bits)**: 127.0.0.1, localhost address.

6 Analysis

Packet Analysis

This packet represents loopback communication for a FastAPI file upload:

- The packet is using the loopback interface (127.0.0.1).
- It's a TCP packet (protocol 6), likely part of the HTTP communication for the file upload.
- The total length (545 bytes) suggests this packet contains a portion of the data being uploaded.
- The Don't Fragment flag is set, which is common for local communications.
- The TTL of 64 is standard for many operating systems for loopback traffic.
- The header checksum is 0, which is normal for loopback interfaces where checksum validation is often disabled.

This packet likely represents a data transfer segment of the file upload process, carrying a portion of the file content.

7 Conclusion

Summary

This analysis demonstrates the structure and content of an IPv4 header from a captured packet during a FastAPI file upload process. The packet shows typical characteristics of loopback communication, with TCP as the transport protocol. The 545-byte total length indicates that this packet is carrying a significant amount of data, likely a chunk of the file being uploaded. To fully analyze the entire file upload, additional packets from the capture would need to be examined to observe the complete data transfer process.