

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:

```

1 from fastapi import FastAPI, UploadFile, File, HTTPException, status
2 import os
3 import uuid
4
5 app = FastAPI()
6
7 UPLOAD_DIRECTORY = "uploads"
8 MAX_FILE_SIZE = 50 * 1024 * 1024 # 50MB
9
10 os.makedirs(UPLOAD_DIRECTORY, exist_ok=True)
11
12 @app.get("/")
13 def home():
14     return {"message": "Welcome to FastAPI File Upload Example"}
15
16 @app.post("/upload/", status_code=status.HTTP_201_CREATED)
17 async def upload_file(file: UploadFile = File(...)):
18     try:
19         file_content = await file.read()
20
21         file_size = len(file_content)
22         if file_size > MAX_FILE_SIZE:
23             raise HTTPException(
24                 status_code=status.HTTP_413_REQUEST_ENTITY_TOO_LARGE,
25                 detail=f"File size exceeds {MAX_FILE_SIZE // (1024 * 1024)}MB
26                     limit",
27             )
28
29         unique_id = str(uuid.uuid4())
30         file_extension = os.path.splitext(file.filename)[-1].lower()
31         uploaded_file_name = f"{unique_id}{file_extension}"
32         file_path = os.path.join(UPLOAD_DIRECTORY, uploaded_file_name)
33
34         with open(file_path, "wb") as buffer:
35             buffer.write(file_content)
36
37         return {"file_name": uploaded_file_name, "file_size": file_size}
38
39     except HTTPException as e:
40         raise e
41     except Exception as e:
42         raise HTTPException(
43             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
44             detail=f"Failed to upload file. Error: {e}",
45         )
46
47 if __name__ == "__main__":
48     import uvicorn
49     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.

6839	15.288389	127.0.0.1	127.0.0.1	HTTP	549	POST /upload/ HTTP/1.1 (application/x-diskcopy)
6840	15.288414	127.0.0.1	127.0.0.1	TCP	56	8000 → 62166 [ACK] Seq=1 Ack=16284809 Win=395456 Len=0 TSval=588043896 TSecr=663164262
6841	15.288425	127.0.0.1	127.0.0.1	TCP	56	8000 → 62166 [ACK] Seq=1 Ack=16366469 Win=313792 Len=0 TSval=588043896 TSecr=663164262
6842	15.288433	127.0.0.1	127.0.0.1	TCP	56	8000 → 62166 [ACK] Seq=1 Ack=16431797 Win=589760 Len=0 TSval=588043896 TSecr=663164262
6843	15.288446	127.0.0.1	127.0.0.1	TCP	56	8000 → 62166 [ACK] Seq=1 Ack=16546121 Win=395456 Len=0 TSval=588043896 TSecr=663164262
6844	15.292771	127.0.0.1	127.0.0.1	TCP	56	8000 → 62166 [ACK] Seq=1 Ack=16546614 Win=896320 Len=0 TSval=588043900 TSecr=663164262
6845	15.295300	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8000 → 62166 [ACK] Seq=1 Ack=16546614 Win=1408320 Len=0 TSval=588043903 TSecr=663164262
6846	15.296270	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8000 → 62166 [ACK] Seq=1 Ack=16546614 Win=1570944 Len=0 TSval=588043903 TSecr=663164262
6877	15.312413	127.0.0.1	127.0.0.1	HTTP/..	263	HTTP/1.1 201 Created , JSON (application/json)
6878	15.312454	127.0.0.1	127.0.0.1	TCP	56	62166 → 8000 [ACK] Seq=16546614 Ack=208 Win=408864 Len=0 TSval=663164286 TSecr=588043920

Figure 1: Packet file-upload

<div> <div> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 </div> <div> <div> 0100 = Version: 4 </div> <div> 0101 = Header Length: 20 bytes (5) </div> <div> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) <div> 0000 00.. = Differentiated Services Codepoint: Default (0) </div> <div>00 = Explicit Congestion Notification: Not ECN-Capable Transport (0) </div> </div> <div> Total Length: 545 </div> <div> Identification: 0x0000 (0) </div> <div> 010. = Flags: 0x2, Don't fragment <div> 0... = Reserved bit: Not set </div> <div> .1.. = Don't fragment: Set </div> <div> ..0. = More fragments: Not set </div> <div> ...0 0000 0000 0000 = Fragment Offset: 0 </div> </div> <div> Time to Live: 64 </div> <div> Protocol: TCP (6) </div> <div> Header Checksum: 0x0000 [validation disabled] <div> [Header checksum status: Unverified] </div> </div> <div> Source Address: 127.0.0.1 </div> <div> Destination Address: 127.0.0.1 </div> </div> </div>
--

Figure 2: Header Fields

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.