Wireshark IP Packet Analysis: FastAPI File Upload

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

July 13, 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, File, UploadFile
from fastapi.responses import HTMLResponse

app = FastAPI()

@app.post("/uploadfile/")
async def create_upload_file(file: UploadFile = File(...)):
    content = await file.read()
    return {"filename": file.filename}

@app.get("/")
async def main():
    content = """
    <html>
        <body>
            <form action="/uploadfile/" enctype="multipart/form-data" method="post">
                <input name="file" type="file">
                <input type="submit">
            </form>
        </body>
    </html>
    """
    return HTMLResponse(content)
```

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.

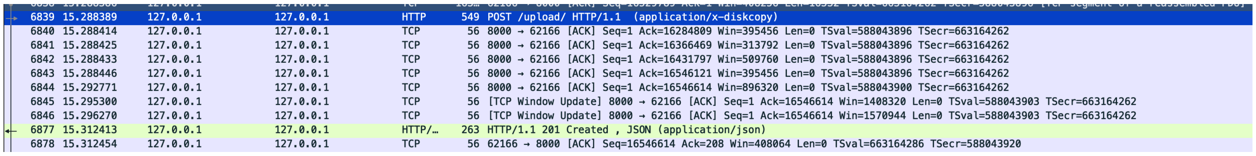
2 Packet Capture Details

Capturing the Packet with Wireshark

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

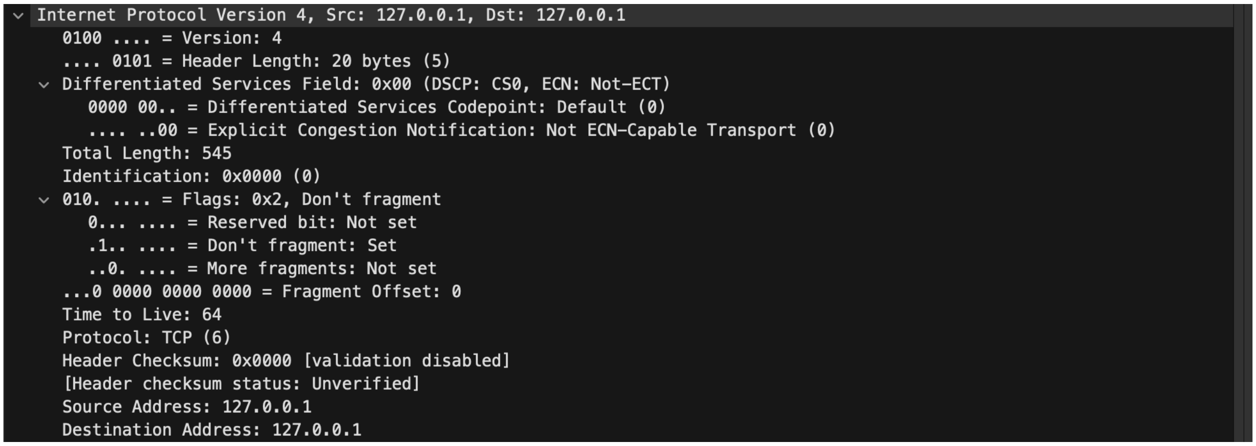
1. Start Wireshark and select the network interface used for localhost communication (e.g., ”Loopback: lo0” on Linux).
2. Begin the packet capture.
3. Perform the file upload to the FastAPI application by accessing the HTML form and submitting a file.
4. Stop the packet capture once the upload is complete.
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.



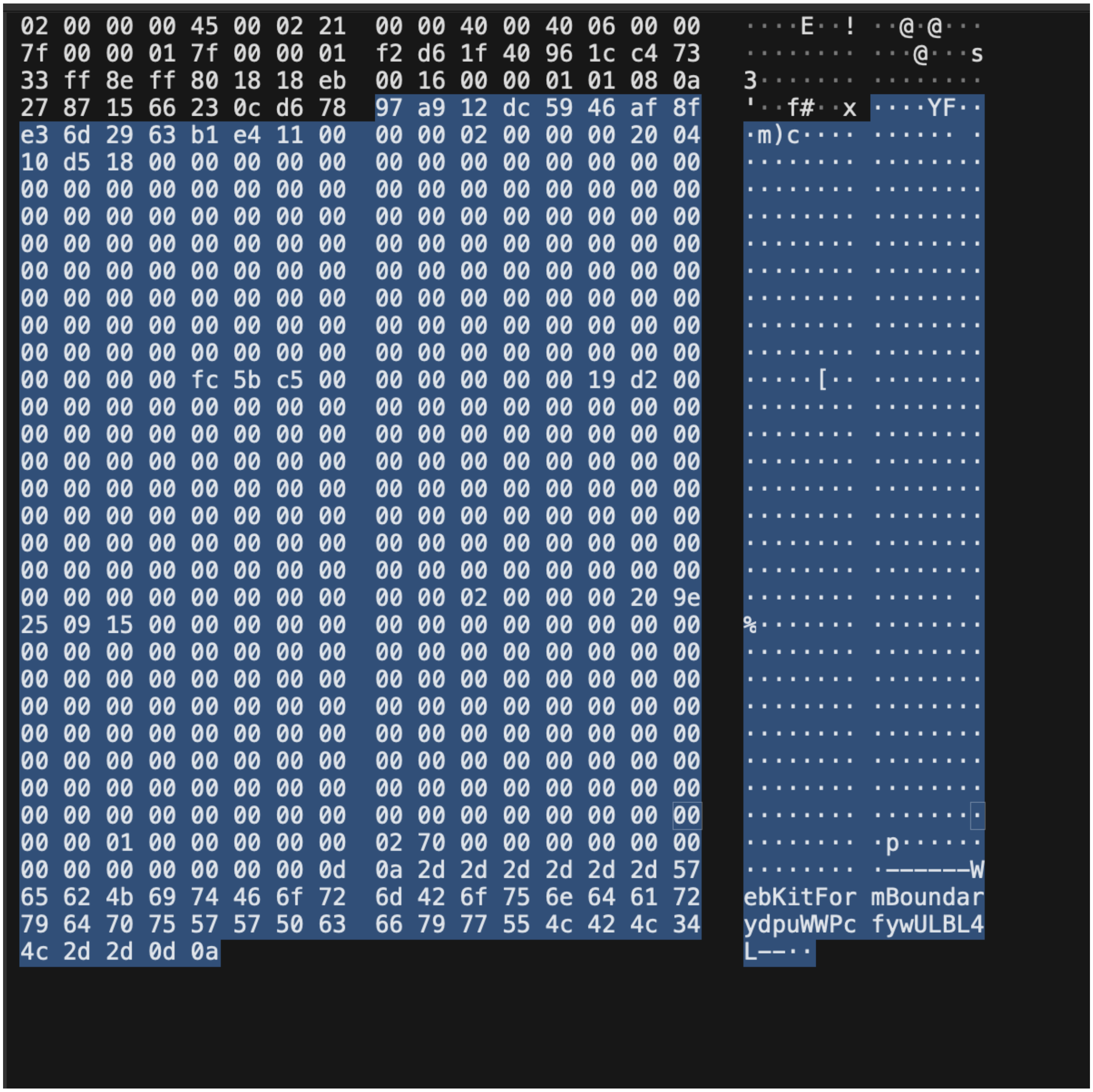
```
6839 15.288389  127.0.0.1  127.0.0.1  HTTP     549 POST /upload/ HTTP/1.1  (application/x-diskcopy)
6840 15.288414  127.0.0.1  127.0.0.1  TCP       56 8000 → 62166 [ACK] Seq=1 Ack=16284809 Win=395456 Len=0 TSval=588043896 TSecr=663164262
6841 15.288425  127.0.0.1  127.0.0.1  TCP       56 8000 → 62166 [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=509760 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=408064 Len=0 TSval=663164286 TSecr=588043920
```

Figure 1: Packet file-upload



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

Figure 2: Header Fields



```
02 00 00 00 45 00 02 21  00 00 40 00 40 06 00 00   ····E··! ··@·@···
7f 00 00 01 7f 00 00 01  f2 d6 1f 40 96 1c c4 73   ········ ···@···s
33 ff 8e ff 80 18 18 eb  00 16 00 00 01 01 08 0a   3······· ········
27 87 15 66 23 0c d6 78  97 a9 12 dc 59 46 af 8f   '··f#··x ····YF··
e3 6d 29 63 b1 e4 11 00  00 00 02 00 00 00 20 04   ·m)c···· ······ ·
10 d5 18 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 fc 5b c5 00  00 00 00 00 00 19 d2 00   ·····[·· ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 02 00 00 00 20 9e   ········ ······ ·
25 09 15 00 00 00 00 00  00 00 00 00 00 00 00 00   %······· ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00 00 01 00 00 00 00 00  02 70 00 00 00 00 00 00   ········ ·p······
00 00 00 00 00 00 00 0d  0a 2d 2d 2d 2d 2d 2d 57   ········ ·------W
65 62 4b 69 74 46 6f 72  6d 42 6f 75 6e 64 61 72   ebKitFor mBoundar
79 64 70 75 57 57 50 63  66 79 77 55 4c 42 4c 34   ydpuWWPc fywULBL4
4c 2d 2d 0d 0a                                     L--··
```

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.