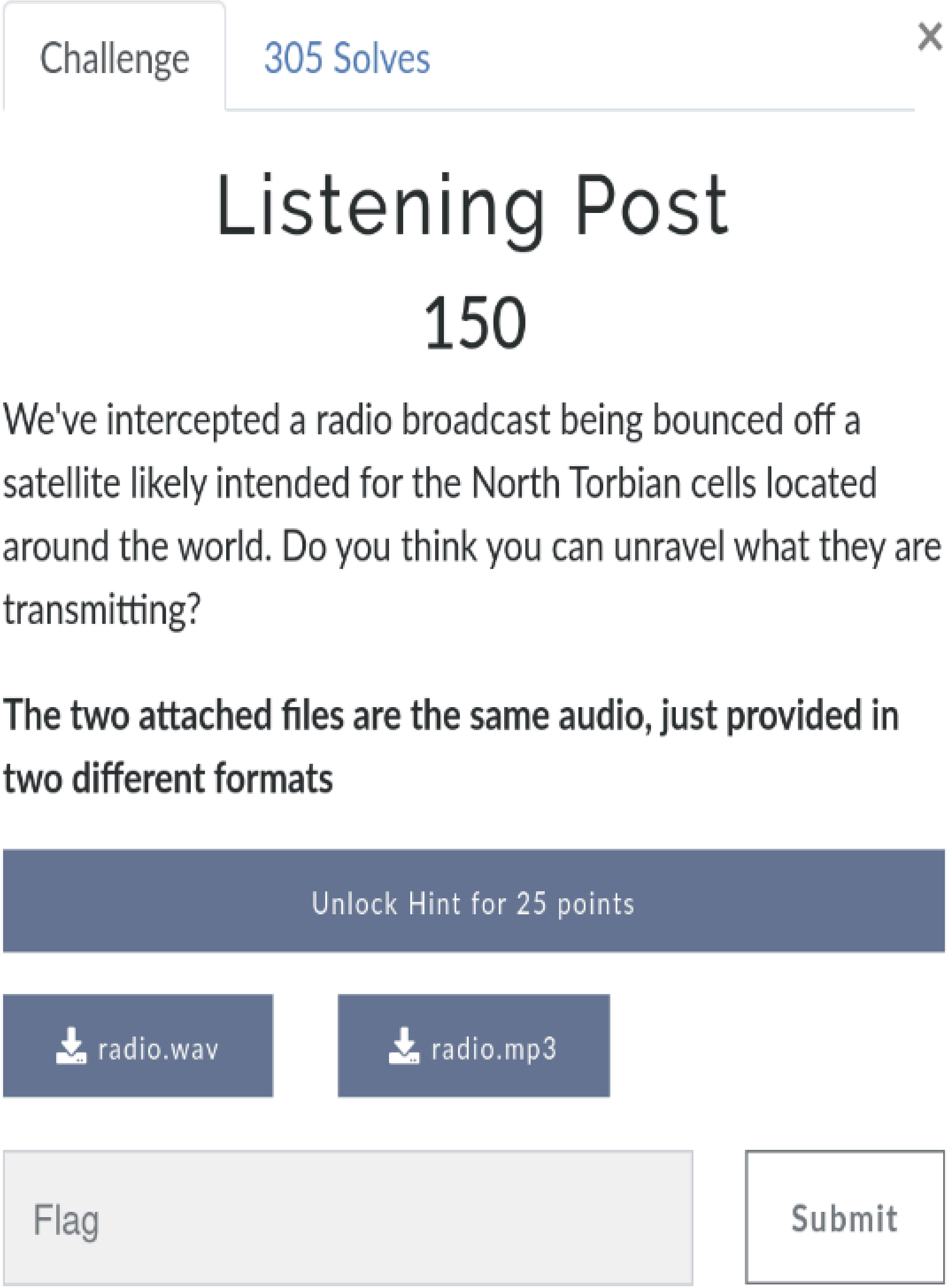
Listening Port

Challenge



Listening Post

150

We've intercepted a radio broadcast being bounced off a satellite likely intended for the North Torbian cells located around the world. Do you think you can unravel what they are transmitting?

The two attached files are the same audio, just provided in two different formats

Workflow

- The challenge provided audio in `.wav` and `.mp3` formats and hinted that it contained a secret transmission.

1. I played the `.wav` file and immediately noticed consistent dual tones, similar to phone dialing sounds — suggesting DTMF encoding (Dual-Tone Multi-Frequency).
2. I used an online tool at dtmf.netlify.app to decode the tones into binary.
3. The site returned a long string of binary that looked like the core of the message

```
This is apparetly the wave decoded in DTMF: 0001s .........000000......1111
0002s 1......000000......000000
0003s ......000000.....000000..
0004s ....111111......111111...
0005s ...000000.....000000.....
0006s .111111......111111......
0007s 000000.....000000......00
0008s 0000......111111......000
0009s 000.....111111......11111
0010s 1......111111......11111.
0011s .....000000......111111..
0012s ....111111......00000....
0013s ..111111......111111.....
0014s .111111......00000......0
0015s 00000......111111......00
0016s 0000.....000000......0000
0017s 00......111111......11111
0018s 1.....000000......111111.
0019s .....000000......000000..
0020s ...000000......111111....
0021s ..111111......000000.....
0022s 000000......111111......0
0023s 00000......000000.....000
0024s 000......000000......1111
0025s 11......11111......000000
0026s ......000000......000000.
0027s .....11111......000000...
0028s ...111111......111111....
0029s .000000......111111......
0030s 111111......111111.....11
```

```
0031s 1111......000000......111
0032s 111......000000.....11111
0033s 1......111111......111111
0034s ......111111.....111111..
0035s ....000000......111111...
0036s ...111111.....000000.....
0037s .111111......000000......
0038s 111111.....111111......00
0039s 0000......000000......111
0040s 11......111111......00000
0041s 0......000000......000000
0042s .....111111......000000..
0043s ....111111......11111....
0044s ..000000......111111.....
0045s .111111.....000000......0
0046s 00000......000000......11
0047s 1111.....111111......0000
0048s 00......111111......11111
0049s 1.....000000......000000.
0050s .....000000......000000..
0051s ...111111......111111....
0052s ..000000......000000.....
0053s 111111......111111......0
0054s 00000......11111......111
0055s 111......000000......0000
0056s 00.....111111......000000
0057s ......000000......000000.
0058s .....11111......000000...
0059s ...111111......111111....
0060s .111111......111111......
0061s 111111......000000.....11
0062s 1111......111111......111
0063s 111......000000.....11111
0064s 1......000000......000000
0065s ......000000.....111111..
0066s ....111111......000000...
0067s ...111111.....000000.....
0068s .000000......000000......
0069s 00000......000000......11
0070s 1111......111111.....0000
```

```
0071s 00......000000......11111
0072s 1......111111.....000000.
0073s .....111111......000000..
0074s ....111111.....111111....
0075s ..111111......111111.....
0076s .111111.....000000......1
0077s 11111......111111......11
0078s 1111.....000000......1111
0079s 11......000000......00000
0080s 0.....000000......000000.
0081s .....111111......111111..
0082s ...000000......000000....
0083s ..000000......00000......
0084s 000000......111111......1
0085s 11111......11111......000
0086s 000......000000......1111
0087s 11.....000000......000000
0088s ......111111......111111.
0089s ....000000......000000...
0090s ...000000......111111....
0091s .000000......000000......
0092s 000000......111111.....11
0093s 1111......000000......000
0094s 000......000000.....11111
0095s 1......000000......111111
0096s ......11111......000000..
0097s ....000000......000000...
0098s ...00000......111111.....
0099s .000000......111111.....0
0100s 00000......111111......11
0101s 1111......111111.....1111
0102s 11......111111......00000
0103s 0......111111.....111111.
0104s .....111111......000000..
0105s ....000000.....111111....
0106s ..111111......000000.....
0107s .111111.....111111......1
0108s 11111......000000......11
0109s 1111.....000000......0000
0110s 00......000000......00000
```

```
0111s ......111111......111111.
0112s .....000000......11111...
0113s ...000000......000000....
0114s ..000000.....111111......
0115s 111111......111111......0
0116s 00000.....000000......111
0117s 111......000000......0000
0118s 00.....111111......111111
0119s ......111111......111111.
0120s ....111111......000000...
0121s
...111110100001100110001011110110111001000110100011001000011000101101111010111
110110101100110001011011000110110000110011011001000101111101110100011010000011
001101011111011101000011000001110010011000100011000101100001010111110111001101
110100001101000111001001111101
```

1. I isolated the clean binary sequence at the end and tried decoding it directly to ASCII — but it was mostly unreadable.
2. I felt it had to be in that last chunk of binary so i copied it into chat gpt and it decoded the rest for me. Here is the thread:

Perfect. The last 4 lines of that dump are **pure binary** — that’s where your flag is hiding.

🔍 What You’re Seeing:

Looks like:

```
...111110100001100110001011110110111001000110100011001000011000101101111010111
110110101100110001011011000110110000110011011001000101111101110100011010000011
001101011111011101000011000001110010011000100011000101100001010111110111001101
110100001101000111001001111101
```

This is likely a **binary ASCII stream**, and we can decode it directly.

✅ Decode the Binary to ASCII (Final Step)

I’ll take that binary and decode it for you now.

```
# Raw binary string from the last few lines
binary_data = (

"11111010000110011000101111011011100100011010001100100001100010110111101011111
011010110011000101101100011011000011001101100100010111110111010001101000001100
110101111101110100001100000111001001100010001100010110000101011111011100110111
0100001101000111001001111101"
)

# Group into 8-bit chunks and convert to ASCII
decoded_chars = [chr(int(binary_data[i:i+8], 2)) for i in range(0,
len(binary_data), 8)]
decoded_message = ''.join(decoded_chars)

decoded_message
```

```
Result:
'ú\x19\x8bÛ\x91£!\x8bzûY\x8bca\x9b"û£A\x9aû¡\x83\x93\x11\x8b\nû\x9b¡£\x93\x1d'
```

The output is **not readable ASCII** — that means the binary isn't straight ASCII but likely **base64-encoded**, XOR'd, or misaligned.

Let's try shifting the binary string by **1 bit**, in case the byte boundary is offset — this is common in CTFs.

I'll try all 8 possible bit shifts and see which one gives us a readable ASCII string.

```
# Try all 8 bit shifts and decode each to ASCII to check for a readable
message
possible_messages = []

for shift in range(8):
    shifted = binary_data[shift:]  # Shift the binary string
    # Ensure the length is a multiple of 8
    trimmed = shifted[:len(shifted) - (len(shifted) % 8)]
    try:
        chars = [chr(int(trimmed[i:i+8], 2)) for i in range(0, len(trimmed),
8)]
        message = ''.join(chars)
        possible_messages.append((shift, message))
    except ValueError:
        continue
```

```
possible_messages
```

```
[(0,
  'ú\x19\x8bÛ\x91£!\x8bzûY\x8bca\x9b"û£A\x9aû¡\x83\x93\x11\x8b\nû\x9b¡£\x93'),
 (1, "ô3\x17·#FC\x16õö³\x16ÆÃ6E÷F\x835÷C\x07&#\x16\x15÷7CG'"),
 (2, 'èf/nF\x8c\x86-ëíf-\x8d\x86l\x8bî\x8d\x06kî\x86\x0eLF,+în\x86\x8eO'),
 (3,

'ÐÌ^Ü\x8d\x19\x0c[×ÚÌ[\x1b\x0cÙ\x17Ý\x1a\x0c×Ý\x0c\x1c\x98\x8cXWÜÝ\r\x1c\x9f')
,
 (4, '¡\x98½¹\x1a2\x18·¯µ\x98¶6\x19²/°4\x19¯°\x1891\x18°¯¹°\x1a9>'),
 (5, 'C1{r4d1o_k1ll3d_th3_t0rb1a_st4r}'),
 (6, '\x86böähÈbÞ¾ÖbØØfÈ¾èÐf¾è`äÄbÂ¾æèhä'),
 (7, '\x0cÅíÈÑ\x90Å½}¬Å±ºÍ\x91}Ñ\xa0Í}ÐÁÉ\x88Å\x85}ÍÐÑÉ')]
```

Boom — on **bit shift 5**, we get a clean decoded flag:

```
C1{r4d1o_k1ll3d_th3_t0rb1a_st4r}
```