c4ptur3-th3-fl4g Walkthrough | TryHackMe

Task 1: Translation and Shifting

CTF challenges often test your ability to recognize and reverse various types of encodings and text transformations. This section walks you through some common techniques used to hide messages — and how to decode them.

Translate, shift and decode the following;

c4n y0u c4p7u23 7h3 f149?
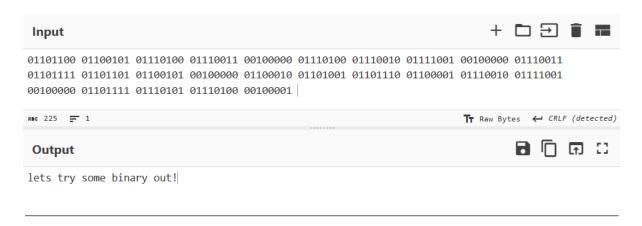
can you capture the flag?

Explanation:
This is written in **leetspeak**, where letters are replaced with similar-looking numbers or characters.

01101100 01100101 01110100 01110011 00100000 01110100 01110010 01111001 00100000 01110011 01101111 01101101 01100101 00100000 01100010 01101001 01101110 01100001 01110010 01111001 00100000 01101111 01110101 01110100 00100001

lets try some binary out!

Explanation:
This is a simple **binary representation of ASCII characters**. Each 8-bit group (byte) corresponds to a letter. Use a binary-to-text converter to decode it.

Input   + ☐ ⊋ 🗑 ▬

```
01101100 01100101 01110100 01110011 00100000 01110100 01110010 01111001 00100000 01110011
01101111 01101101 01100101 00100000 01100010 01101001 01101110 01100001 01110010 01111001
00100000 01101111 01110101 01110100 00100001 |
```

ᴀʙᴄ 225 ☰ 1   Tr Raw Bytes ↩ CRLF (detected)

Output   💾 🗐 🔁 ⛶

lets try some binary out!|

MJQXGZJTGIQGS4ZAON2XAZLSEBRW63LNN5XCA2LOEBBVIRRHOM======

Explanation:

Base32 encodes binary data using a 32-character set (A–Z and 2–7). It's often used in CTFs because it's URL-safe and not as obvious as Base64.

Input   + ☐ ⤷ 🗑 ▬

MJQXGZJTGIQGS4ZAON2XAZLSEBRW63LNN5XCA2LOEBBVIRRHOM======|

ᴬᴮᶜ 56   ☰ 1   Tᴛ Raw Bytes   ← CRLF (detected)

Output   🖫 🗍 🗗 ⌂

base32 is super common in CTF's

RWFjaCBCYXNlNjQgZGlnaXQgcmVwcmVzZW50cyBleGFjdGx5IDYgYml0cyBvZiBkYXRhLg==

Explanation:

Base64 represents binary data using 64 printable ASCII characters. It's easy to spot due to the use of = padding and common in file/email encodings.
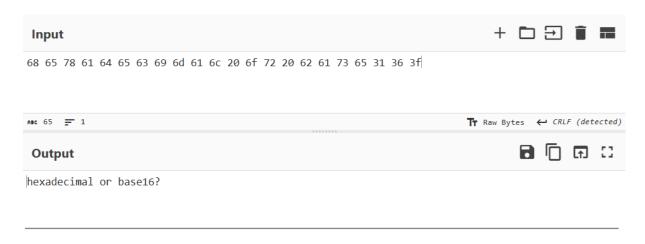
Input   + ☐ ⤷ 🗑 ▬

RWFjaCBCYXNlNjQgZGlnaXQgcmVwcmVzZW50cyBleGFjdGx5IDYgYml0cyBvZiBkYXRhLg==|

ᴬᴮᶜ 72   ☰ 1   Tᴛ Raw Bytes   ← CRLF (detected)

Output   🖫 🗍 🗗 ⌂

Each Base64 digit represents exactly 6 bits of data.|

---

68 65 78 61 64 65 63 69 6d 61 6c 20 6f 72 20 62 61 73 65 31 36 3f

Explanation:

Each pair of hex digits represents one ASCII character. This is **base16 encoding**, and it's easily reversible using any hex-to-text tool.

Input   + 🗁 ⇥ 🗑 ▤

```
68 65 78 61 64 65 63 69 6d 61 6c 20 6f 72 20 62 61 73 65 31 36 3f
```

ᴀʙᴄ 65   ☰ 1   Tᴛ Raw Bytes   ↵ CRLF (detected)

Output   💾 🗍 🔼 ⛶

```
hexadecimal or base16?
```

Ebgngr zr 13 cynprf!

Explanation:

ROT13 is a Caesar cipher that shifts each letter 13 positions in the alphabet. It's symmetrical — applying ROT13 twice gives you the original text.

Input   + 🗁 ⇥ 🗑 ▤

```
Ebgngr zr 13 cynprf!
```

ᴀʙᴄ 22   ☰ 2   Tᴛ Raw Bytes   ↵ CRLF (detected)

Output   💾 🗍 🔼 ⛶

```
Rotate me 13 places!
```
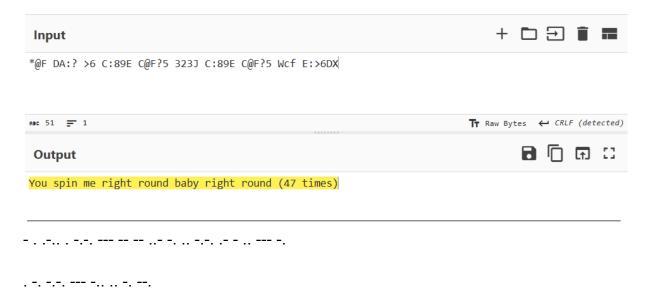
*@F DA:? >6 C:89E C@F?5 323J C:89E C@F?5 Wcf E:>6DX

Explanation:

ROT47 is similar to ROT13 but includes all printable ASCII characters. Each character is rotated by 47 positions in the ASCII table.

You spin me right round baby right round (47 times)

Input

```
*@F DA:? >6 C:89E C@F?5 323J C:89E C@F?5 Wcf E:>6DX
```

ABC 51 = 1   Tr Raw Bytes ← CRLF (detected)

Output

You spin me right round baby right round (47 times)

- . .-.. . -.-. --- -- -- ..- -. .. -.-. .- - .. --- -.

. -. -.-. --- -.. .. -. --.

TELECOMMUNICATIO ENCODING

Explanation:
 This is **Morse code**, which encodes text using dots and dashes. It's common in puzzle-based challenges. Slashes or spaces separate characters/words.

Input

```
- . .-.. . -.-. --- -- -- ..- -. .. -.-. .- - .. --- -.

. -. -.-. --- -.. .. -. --.
```

ABC 86 = 3   Tr Raw Bytes ← CRLF (detected)

Output

TELECOMMUNICATIO ENCODING

---

85 110 112 97 99 107 32 116 104 105 115 32 66 67 68

Unpack this BCD

Explanation:
 This is actually **ASCII** (not strict BCD), where each number maps to a character. However, it plays on the idea of breaking data into nibbles (4 bits).

Input

```
85 110 112 97 99 107 32 116 104 105 115 32 66 67 68
```

ABC 51 = 1   Tr Raw Bytes ← CRLF (detected)

Output

Unpack this BCD

LS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0KLS0tLS0
gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLi0tLS0KLS0tLS0gLS0tLS
0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS
0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLS0tLS0gLS0g
Li0tLS0gLi0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0gLS0tLS0gLS0gL
S0tLS0gLS0tLS0gLS0tLS0gLS0gLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0t
LS0tLS0gLS0tLS0gLS0tLS0gLS0gLS0tLS0gLS0tLS0gLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0t
LS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gL
S0tLS0gLi0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0t
LS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0gLS0KLS0tLS0gLi0tL
S0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gL
S0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0t
LS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gL
i0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLi0tLS0gLi0t
LS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS
0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KL
S0tLS0gLi0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0KLS0tLS0gLi0tLS0gL
i0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLS0tLS0gLi0t
LS0gLi0tLS0gLi0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0
gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0t
LS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0KLS0t
LS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gL
S0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLi0t
LS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gL
S0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0KL
S0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0KL

S0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0gLS0tLS0gLS0gLS0tLS0gLS0tLS0KLS0tLS
0gLi0tLS0gLi0tLS0gLS0tLS0gLS0gLS0gLS0gLS0gLS0gLS0gLS0KLS0tLS0gLi0t
LS0gLS0tLS0gLi0tLS0gLi0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tL
S0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0gLS
0tLS0gLS0tLS0gLS0tLS0gLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0g
LS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0g
LS0gLS0tLS0gLS0tLS0gLS0gLS0gLi0
tLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0gLS0
LS0tLS0gLS0tLS0KLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS
0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0t
LS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0
LS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0KLS0tLS
0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0KLS0gLS0tL
S0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0gLi0tLS0gLi0
tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0KLS0gLi0tLS0gLi0tLS0gL
S0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0KLS0gLS0tLS
0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLi0t
LS0gLS0tLS0gLS0tLS0gLS0tLS0gLi
0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0
LS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0g
Li0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS
0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0t
LS0gLS0tLS0gLS0tLS0gLi0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS
0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0g
LS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0KLS0tLS0gLS0t
LS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tL
S0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS
0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0t
LS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLS0tLS0gLi0tLS
0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0
gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0t
LS0gLi0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0t
LS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLi0tLS
0gLS0tLS0gLS0tLS0gLS0tLS0KLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tL
S0gLS0tLS0KLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS
0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0

tLS0gLi0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0KLi0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLi0tLS0gLi0tLS0gLi0tLS0gLi0KLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLi0tLS0=

Base64 -> Morse Code -> Binary -> ROT47 -> Decimal
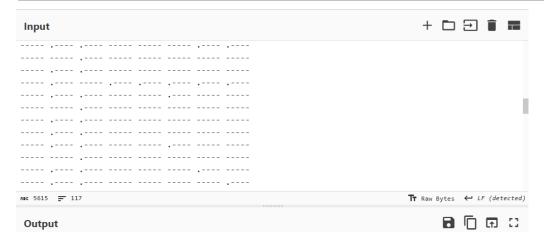
Explanation:
Some challenges use **layered encoding** to increase complexity. In this case, you must:

1. Decode from **Base64**
2. Convert Morse code to **text**
3. Translate that to **binary**
4. Apply **ROT47**
5. Finally interpret as **decimal**

Input

0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0K
LS0tLS0gLi0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS
0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0t
LS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS
0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLS0tLS0gLi0tLS0g
LS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0t
LS0gLi0tLS0KLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLS0t
LS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLS0tLS0KLS0tLS0gLi0tLS0gLi0tLS0gLS0tLS0gLS0tLS0gLS0tLS0gLi0tLS0gLi0tLS0KLS0tLS0gLi0tLS0gLi0tLS
0gLS0tLS0gLS0tLS0gLi0tLS0gLS0tLS0gLi0tLS0=

ABC 7490   ═ 2   Tr Raw Bytes   ← CRLF (detected)

Output ✗ˣ

----- .---- .---- ----- ----- .---- .---- -----
----- .---- .---- ----- ----- .---- ----- .----
----- ----- .---- ----- ----- ----- ----- -----
----- .---- .---- ----- .---- ----- ----- -----
----- .---- .---- .---- .---- .---- ----- -----
----- .---- .---- ----- .---- ----- ----- -----
----- ----- .---- ----- ----- ----- ----- -----
----- .---- .---- ----- ----- ----- ----- -----
----- .---- .---- ----- .---- ----- ----- -----
----- .---- .---- ----- ----- .---- ----- .----
----- .---- .---- ----- ----- ----- ----- -----
----- .---- ----- ----- ----- ----- ----- -----
----- ----- .---- ----- .---- ----- ----- -----
----- ----- .---- ----- ----- ----- ----- -----

ABC 5615   ═ 117   ⏱ 6ms   Tr Raw Bytes   ← LF (detected)

Input

----- .---- .---- ----- ----- ----- .---- .----
----- ----- .---- ----- ----- ----- ----- -----
----- .---- .---- ----- ----- ----- ----- -----
----- .---- ----- .---- .---- .---- .---- .----
----- .---- .---- ----- ----- .---- ----- -----
----- ----- .---- ----- ----- ----- ----- -----
----- .---- .---- ----- ----- ----- .---- -----
----- .---- .---- ----- ----- ----- ----- -----
----- .---- .---- ----- ----- .---- ----- -----
----- ----- .---- ----- ----- ----- ----- -----
----- .---- ----- ----- ----- ----- .---- -----
----- .---- .---- ----- ----- ----- ----- .----

ABC 5615   ═ 117   Tr Raw Bytes   ← LF (detected)

Output

01100110 01100101 00100000 01100000 01011111 01100000 00100000 01100000 01100000 01100101 00100000
01100010 01101000 00100000 01100000 01100000 01100100 00100000 01100010 01100001 00100000 01100000
01011111 01101000 00100000 01101000 01100110 00100000 01100000 01011111 01100110 00100000 01100000
01011111 01100000 00100000 01100010 01100001 00100000 01100000 01100000 01100101 00100000 01100000
01011111 01100011 00100000 01011111 01100100 00100000 01100000 01100000 01100000 01100000 00100000
01100010 01100001 00100000 01101000 01100110 00100000 01100010 01100001 00100000 01101000 01100111
00100000 01100000 01011111 01100100 00100000 01100000 01100000 01100101 00100000 01100010 01100001
00100000 01100000 01100000 01100101 00100000 01100000 01100000 01100011 00100000 01100000 01011111
01100100 00100000 01101000 01101000 00100000 01100000 01011111 01100110 00100000 01100000 01011111
01100100 00100000 01100000 01011111 01100000 00100000 01100000 01100000 01100011 00100000 01100011
01100101 00100000 01100011 01100101 00100000 01100011 01100101

ABC 1052   ═ 1   ⏱ 1ms   Tr Raw Bytes   ← LF (detected)

```
01100110 01100101 00100000 01100000 01011111 01100000 00100000 01100000 01100000 01100101 00100000
01100010 01101000 00100000 01100000 01100000 01100100 00100000 01100010 01100001 00100000 01100000
01011111 01101000 00100000 01101000 01100110 00100000 01100000 01011111 01100110 00100000 01100000
01011111 01100000 00100000 01100010 01100001 00100000 01100000 01100000 01100101 00100000 01100000
01011111 01100011 00100000 01100000 01011111 01100100 00100000 01100000 01100000 01100100 00100000
01100010 01100001 00100000 01101000 01100110 00100000 01100010 01100001 00100000 01101000 01100111
00100000 01100000 01011111 01100100 00100000 01100000 01100000 01100101 00100000 01100010 01100001
00100000 01100000 01100000 01100101 00100000 01100000 01100000 01100011 00100000 01100000 01011111
01100100 00100000 01101000 01101000 00100000 01100000 01011111 01100110 00100000 01100000 01011111
01100100 00100000 01100000 01011111 01100000 00100000 01100000 01100000 01100011 00100000 01100011
01100101 00100000 01100011 01100101 00100000 01100011 01100101
```

ABC 1053 — 1   Tr Raw Bytes   ← LF (detected)

Output 💾 ⧉ ⬑ ⤢

```
fe `_` ``e bh ``d ba `_h hf `_f `_` ba ``e `_c `_d ``d ba hf ba hg `_d ``e ba ``e ``c `_d hh `_f `_d
`_` ``c ce ce ce
```

Input + ▭ ⭲ 🗑 ▬

```
fe `_` ``e bh ``d ba `_h hf `_f `_` ba ``e `_c `_d ``d ba hf ba hg `_d ``e ba ``e ``c `_d hh `_f `_d
`_` ``c ce ce ce
```

ABC 117 — 1   Tr Raw Bytes   ← LF (detected)

Output ✎ˣ 💾 ⧉ ⬑ ⤢

76 101 116 39 115 32 109 97 107 101 32 116 104 105 115 32 97 32 98 105 116 32 116 114 105 99 107 105
101 114 46 46 46

Input + ▭ ⭲ 🗑 ▬

```
76 101 116 39 115 32 109 97 107 101 32 116 104 105 115 32 97 32 98 105 116 32 116 114 105 99 107 105
101 114 46 46 46
```

ABC 117 — 1   Tr Raw Bytes   ← LF (detected)

Output 💾 ⧉ ⬑ ⤢

Let's make this a bit trickier...

Task 2: Spectograms

A **spectrogram** is a visual representation of the **frequency content of a sound** over time. It shows how the **pitch (frequency)** and **intensity (amplitude)** of a signal change.

- **X-axis:** Time
- **Y-axis:** Frequency
- **Color/Intensity:** Volume (how loud that frequency is)
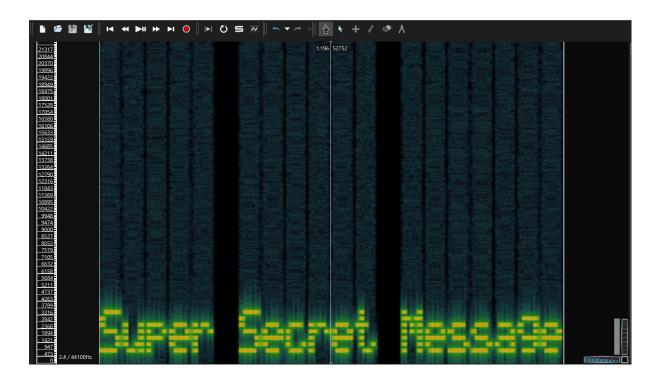
Spectrograms are useful in CTF challenges to **hide or reveal messages in audio files**, since images of spectrograms can contain readable text or QR codes when analyzed with special tools.

Here, we download the file present at the right side and try to gain a message from the file. We found out that the file is a voice note, so we had to try to decode it.

I used Sonic Visualizer to decode the file. We open the file in the sonic visualizer and then after adding that Spectogram, we got our message.

Task 3: Steganography
Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. **Steganography** is the practice of hiding a secret message inside another file — such as an image, audio, video, or text — in a way that doesn't attract attention.

Examples include:

- Hiding text in the **least significant bits (LSB)** of an image.
- Embedding files inside media with tools like steghide or zsteg.
- Concealing messages in whitespace, file metadata, or even pixel color variations.

It differs from encryption: **encryption scrambles the message**, while **steganography hides its existence**.

SpaghettiSteg

```
┌──(user⊛user)-[~]
└─$ steghide extract -sf ./Desktop/stegosteg_1559008553457.jpg
Enter passphrase:
wrote extracted data to "steganopayload2248.txt".

┌──(user⊛user)-[~]
└─$ cat steganopayload2248.txt
SpaghettiSteg
```

Task 4: Security through Obscurity

Security through obscurity is the reliance in security engineering on the secrecy of the design or implementation as the main method of providing security for a system or component of a system.

Download and get 'inside' the file. What is the first filename & extension?
hackerchat.png

Get inside the archive and inspect the file carefully. Find the hidden text.
AHH_YOU_FOUND_ME!

I found the answer to both the questions by just one thing:
cat <filename of the image>

Summary

In this walkthrough, we explored several common Capture The Flag (CTF) techniques used to hide or obfuscate information, and successfully decoded each challenge:

1. **Translation and Shifting (Task 1):**
   We worked through various encoding formats such as binary, base32, base64, hexadecimal, ROT13, ROT47, and Morse code. Each question required identifying the encoding and decoding it to retrieve the hidden message.
2. **Spectrogram Analysis (Task 2):**
   Using an audio file and the **Sonic Visualizer** tool, we applied a spectrogram filter to reveal a **hidden visual message** embedded in the frequency patterns of the audio:
   "Super Secret Message"
3. **Steganography (Task 3):**
   We looked at techniques for hiding messages inside image and audio files using tools like `steghide` and `zsteg`. This task emphasized how steganography conceals **the existence** of data rather than just protecting it.
4. **Security Through Obscurity (Task 4):**
   By examining an image file (`hackerchat.png`), we uncovered hidden content by simply reading the file with `cat`. This revealed a hidden string and showed how seemingly harmless files can contain embedded data.

I hope you enjoyed this one! Happy Hacking!