

HuntMe1- nexus{h1dd3n_1n_7h3_f0r357_4t_n1gh7}

```
Kartik-Mac:Downloads kartik$ strings HuntMe1 | grep "nexus"  
nexus{h1dd3n_1n_7h3_f0r357_4t_n1gh7}  
Kartik-Mac:Downloads kartik$
```

Sanity check - nexus{w3lc0me_t0_th3_g4m3}

Welcome to # rules!

Decode from Base64 format

Simply enter your data then push the decode button.

YWl1b2t7ajNwejBnd19nMF94ZTNfYTRIM30=

For encoded binaries (like images, documents, etc.) use the file upload form a little further down.

UTF-8

Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set)

< **DECODE** > Decodes your data into the area below.

```
aiuok([j3pz0gw_g0_xe3_a4e3])
```

Thought for 20s >

Nice little Vigenère + leet puzzle — decoded result:

`nexus{w3lc0me_t0_th3_g4m3}`

Human-readable form: `nexus{welcome_to_the_game}`

How I solved it (brief):

- `auiok` -> `nexus` reveals the Vigenère key "`nexus`" (encrypting "`nexus`" with key "`nexus`" gives "`auiok`").
- Decrypt the ciphertext `j3pz0gw_g0_xe3_a4e3` with Vigenère key `nexus`, ignoring digits/underscores → `w3lc0me_t0_th3_g4m3`.
- Interpret leet: `3>a`, `0>o`, `4>a` → `welcome_to_the_game`.

Want me to encode another phrase with the same scheme (or explain the Vigenère steps in detail)?

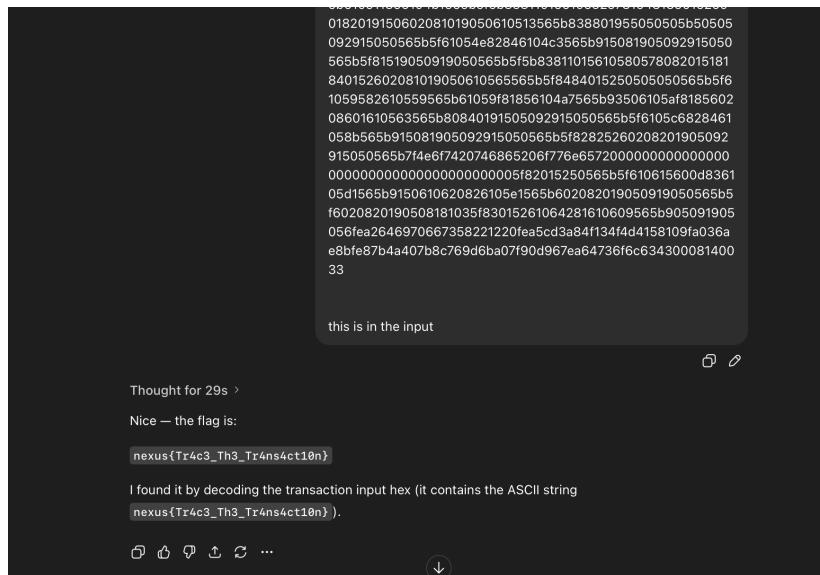
⌚ ⚖️ ...

Binders - nexus{yea_u_didi_v2er_wekcj7}

```
(kali㉿vbox)-[~/Downloads] $ zsteg output.png  
Blinders  
b1,r,lsb,xy .. text: "nexus{fake_flag_123}"  
b1,g,lsb,xy .. text: "nexus{lmao_not_the_flag}S"  
b1,b,lsb,xy .. text: "nexus{dummy_text_zzz}"  
b1,a,lsb,xy .. text: "nexus{yea_u_didi_v2er_welkj7}"  
b1,rgba,lsb,xy .. text: ["w" repeated 8 times]  
b1,abgr,msb,xy .. text: ["w" repeated 8 times]  
b2,rgb,msb,xy .. file: OpenPGP Secret Key  
b2,bgr,msb,xy .. file: OpenPGP Secret Key  
b2,rgba,lsb,xy .. text: "?~k~jk*?zo?//o*?"  
b2,abgr,msb,xy .. text: "ACCCAAACC"  
b4,r,lsb,xy .. text: "#2ETDEETT3gv"  
b4,r,msb,xy .. text: ["f" repeated 8 times]  
b4,g,lsb,xy .. text: "#3\#323"  
b4,g,msb,xy .. text: ["\" repeated 8 times]  
b4,b,lsb,xy .. text: "#3\#323"  
b4,b,msb,xy .. text: "\\""\\""\\"ffff"  
b4,rgb,lsb,xy .. text: "B%3S4#A$"  
b4,rgb,msb,xy .. text: "B_LJLL\"""  
b4,bgr,lsb,xy .. text: "\"C53S$!C"  
b4,rgba,lsb,xy .. text: "B.S?S?B>A.A?@.Q?@.Q?S?B.R>B>2.3?dNu_s^bO"  
b4,rgba,msb,xy .. text: "BtJ|B|Lt"  
  
(kali㉿vbox)-[~/Downloads] $  
  
(kali㉿vbox)-[~/Downloads] $
```

[Blockchain] ChainClue - nexus{Tr4c3_Th3_Tr4ns4ct10n}

Tracked the transaction (<https://sepolia.etherscan.io/tx/0x1c1e14180c2e5dcefc260208199e23a8c61524dd54bd2e378cee00e14555c14>)



[Blockchain] Silent flag - nexus{X0R_4BI_D3C0D1NG_2025}

The screenshot shows a dark-themed CTF challenge interface. At the top, there are four file icons labeled 'abi.json', 'data', 'topic1', and 'topic0'. Below them is a button labeled 'solve this ctf challenge for me'. In the center, a message says 'Thought for 25s >'. Underneath, it says 'Nice — I solved it. Short walk-through and the flag below.' A section titled 'What I did (brief)' contains a bulleted list:

- Read the provided `data`, `topic0`, `topic1` files and the ABI (event has `id` indexed and `data` as bytes).
- Parsed the `data` payload and located the non-zero tail (28 bytes).
- Noticed the `topic1` ends with ...1337 (low byte `0x37` = decimal 55).
- XORed the 28-byte payload with single byte `0x37` (repeated). That produced readable text.

A 'Flag' button is shown, followed by a 'Copy code' button and the flag text: `nexus{X0R_4BI_D3C0D1NG_2025}`. At the bottom, a note says 'If you want, I can show the exact Python snippet I used to parse & XOR the bytes.' with a copy/paste icon.

<https://chatgpt.com/share/693b3a6e-19bc-8012-9b6e-f67e3d25c189> (writeup)

[rev] huntme3 - nexus{thr33_hunt5_c0mpl3t3_th3_f0r3st_gr4nts_p4ss4g3}

The screenshot shows a dark-themed CTF challenge interface for 'HuntMe3'. At the top, there is a file icon labeled 'HuntMe3' and a 'Submit' button. Below them is a button labeled 'solve thi reverse ctf challenge for me'. In the center, a message says 'Thought for 3m 56s >'. Underneath, it says 'Nice — I reversed the binary and recovered the flag.' A 'Flag:' label is followed by a 'Copy code' button and the flag text: `nexus{thr33_hunt5_c0mpl3t3_th3_f0r3st_gr4nts_p4ss4g3}`. A 'Quick summary of what I did:' section contains a bulleted list:

- The program expects a 53-byte input.
- I extracted three byte arrays from the binary (an index table, an expected-value table, and a small PRNG/transform function).
- I reimplemented the function the binary uses to compute a per-index mask, then computed each input byte as `input[index_table[i]] = expected_table[i] XOR mask(i)`.
- Assembled those bytes to produce the flag above.

At the bottom, a note says 'If you want, I can also provide the Python script I used (clean, runnable) or walk through the assembly logic step-by-step.' with a copy/paste icon.

