Success challenge(fully solved)

Challenge Summary

I was given a Haskell program that:

- Requires a flag exactly 39 characters long.
- Converts it to a list of ASCII integer codes.
- Applies a long list of arithmetic and bitwise constraints on selected character positions.

Example constraints include:

- Products: `chars[37] * chars[15] == 3366`
- Sums: `chars[8] + chars[21] == 197`
- Bitwise: `.|.`, `.&.`, `xor`

The goal was to find a single valid flag that satisfies all of these conditions.

My Approach

Since solving dozens of mixed arithmetic and bitwise constraints by hand is practically impossible, I learned to use the Z3 SMT solver for this challenge.

How I Solved It with Z3

Steps I took:

- Studied Z3 basics (how to declare variables, add constraints, and handle bitwise operations).
- Modeled each flag character as a symbolic integer in Z3.
- Restricted each one to be printable ASCII (32 to 126).
- Converted the given Haskell constraints into equivalent Z3 syntax.
    - Learned that Z3 needs BitOr, BitAnd, BitXor instead of `|`, `&`, `^`.

Ran Z3 to find one valid solution.

Final Z3 Solver Code

```
from z3 import *
```

```
# 39 symbolic characters
chars = [Int(f'c{i}') for i in range(39)]

s = Solver()

# Restrict to printable ASCII
for c in chars:
    s.add(c >= 32, c <= 126)

# Constraints (translated from Haskell)
s.add(chars[37] * chars[15] == 3366)
s.add(chars[8] + chars[21] == 197)
s.add(chars[8] * chars[13] == 9215)
s.add(chars[0] * chars[3] == 2714)
s.add(chars[3] + chars[21] == 159)
s.add(chars[1] * chars[20] == 5723)
s.add(BitOr(chars[6], chars[37]) == 105)
s.add(chars[11] * chars[7] == 11990)
s.add(BitAnd(chars[29], chars[25]) == 100)
s.add(BitOr(chars[16], chars[29]) == 127)
s.add(chars[20] - chars[6] == -8)
s.add(chars[21] + chars[20] == 197)
s.add(chars[2] + chars[36] == 77)
s.add(chars[35] * chars[11] == 3630)
s.add(chars[4] * chars[3] == 2714)
s.add(BitXor(chars[35], chars[6]) == 72)
s.add(chars[25] + chars[24] == 221)
s.add(chars[14] * chars[36] == 3465)
s.add((chars[15] - chars[11]) - 148 == -156)
s.add(chars[37] + chars[17] == 138)
s.add(BitXor(chars[1], chars[38]) == 70)
s.add(chars[9] + chars[29] == 212)
s.add(chars[30] - chars[10] == 7)
s.add(chars[10] + chars[33] == 206)
s.add(chars[7] * chars[15] == 11118)
s.add((chars[28] * chars[14]) * 55 == 641025)
s.add(BitOr(BitOr(chars[7], chars[4]), 216) == 255)
s.add(chars[24] + chars[4] == 151)
s.add(chars[2] * chars[30] == 4928)
s.add(chars[5] + chars[22] == 224)
s.add(BitOr(chars[18], chars[36]) == 127)
s.add(chars[13] + chars[34] == 195)
s.add(BitOr(chars[9], chars[17]) == 111)
s.add(chars[12] * chars[9] == 10403)
s.add(BitXor(chars[25], chars[27]) == 23)
s.add(BitXor(chars[13], chars[34]) == 59)
s.add(chars[18] + chars[31] == 200)
s.add(chars[17] + chars[32] == 213)
s.add(chars[2] * chars[12] == 4444)
s.add(chars[24] * chars[31] == 11025)
s.add(chars[5] * chars[0] == 5658)
s.add(chars[10] + chars[32] + 228 == 441)
s.add(chars[35] * chars[0] == 1518)
s.add(BitOr(chars[30], chars[8]) == 113)
s.add(chars[28] - chars[34] == 11)
s.add(chars[26] * chars[14] == 9975)
s.add(chars[31] * chars[22] == 10605)
s.add((chars[26] * chars[32]) * 239 == 2452140)
s.add(chars[28] * chars[38] == 13875)
s.add(chars[18] + chars[16] == 190)
s.add(chars[27] + chars[26] + 96 == 290)
```

```
s.add(chars[22] - chars[38] == -24)
s.add(chars[33] + chars[5] == 224)
s.add(chars[19] * chars[16] == 10355)
s.add(chars[27] + chars[1] == 158)
s.add(chars[33] + chars[12] == 202)
s.add(chars[19] * chars[23] == 10355)

# Solve
if s.check() == sat:
    m = s.model()
    flag = ''.join(chr(m[c].as_long()) for c in chars)
    print(f"Solution: {flag}")
else:
    print("No solution found.")
```

Result

Running this gave me the exact flag that passes all constraints.
More importantly, I learned how to use Z3 — now I can apply it to any similar CTF or reverse-engineering problem.

Key Takeaway

I learned Z3 from scratch specifically for this challenge — it turned out to be the perfect tool for systematically solving a mix of arithmetic and bitwise constraints.