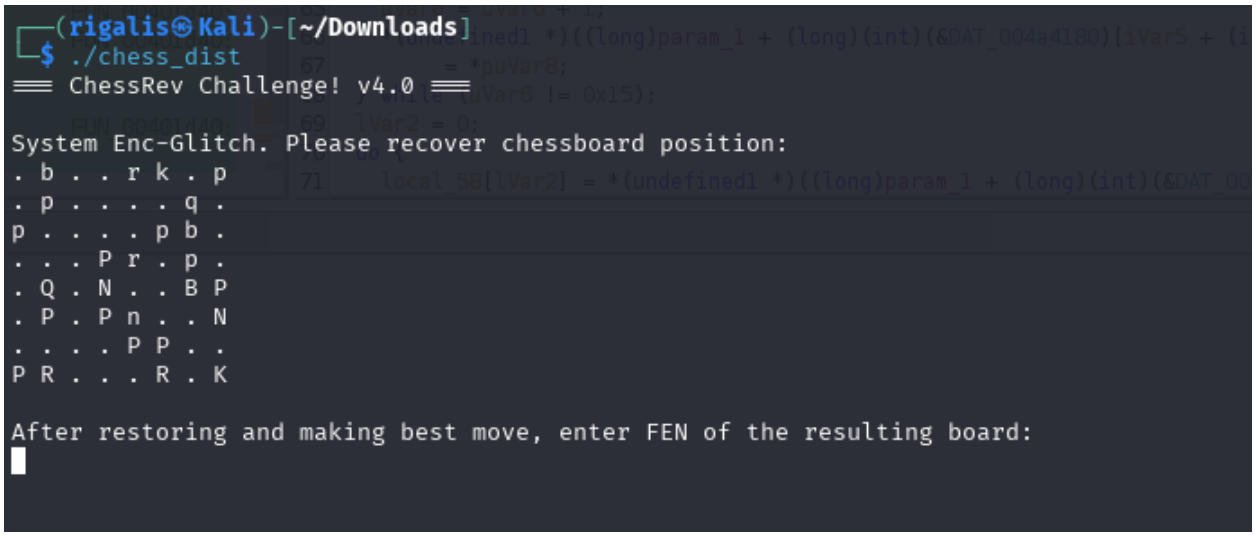
Challenge Name: Glitchy Endgame

Description: *Good news: The chessboard shows up. Bad news: the position isn't playing by anyone's rules.*

Call it experimental board logic or creative data shuffling, the setup challenges standard chess thinking. From code wizardry to chess mastery, whichever works, as long as you restore order and land the winning move.

In this challenge, you are given an ELF binary. Upon execution, the binary displays a scrambled chessboard and prompts for the FEN (Forsyth-Edwards Notation) of the "restored and solved" position.



Initial Recon

Step 1: Identifying the Entry Point

Loading the binary into Ghidra and searching for distinctive strings, we quickly locate the main function by searching for "ChessRev Challenge" or "chess". This leads us to FUN_004016a0, which we can confidently identify as main.

```
undefined8 FUN_004016a0(void)

{
  int iVar1;
  long lVar2;
  code *pcVar3;
  undefined1 *puVar4;
  undefined1 local_147 [15];
  undefined1 local_138 [80];
  undefined1 local_e8 [80];
  undefined1 local_98 [136];

  FUN_004050e0("=== %s ===\n\n","ChessRev Challenge! v4.0");
  FUN_00401c20();
  iVar1 = FUN_004019c0(10000);
  if (iVar1 < 0) {
    FUN_0040add0("Unexpected error.");
  }
  else {
    FUN_00401bb0(local_138);
    FUN_00401d40(local_e8);
    FUN_0040add0("System Enc-Glitch. Please recover chessboard position:");
    FUN_00401f80(local_e8);
    FUN_0040add0("\nAfter restoring and making best move, enter FEN of the resulting board:");
    lVar2 = FUN_0040aa50(local_98,0x80,PTR_DAT_004a4858);
    if (lVar2 == 0) {
      FUN_0040add0("Input error.");
    }
    else {
      lVar2 = thunk_FUN_00417b30(local_98,&DAT_00477b39);
      local_98[lVar2] = 0;
      pcVar3 = (code *)FUN_00420fb0(0,0x2f,7,0x22,0xffffffff,0);
      if (pcVar3 != (code *)0xffffffffffffffff) {
        puVar4 = local_147;
        FUN_00401b10(puVar4);
        FUN_00401b40(pcVar3,&DAT_004a4100,0x2f,puVar4);
        iVar1 = (*pcVar3)(local_98,&DAT_004a5ca0);
        if (iVar1 == 0) {
          FUN_0040add0("Incorrect FEN. Try again.");
        }
        else {
          FUN_0040add0("Correct! You solved it.");
          FUN_00401ff0();
        }
        FUN_00421090(pcVar3,0x2f);
```

Upon inspecting strings and function cross-references in a disassembler, you can quickly locate the main challenge routine. Key functions are observed in code or pseudocode as:

- FUN_00401bb0: Handles board assembly and obfuscation setup.
- FUN_00401d40: Applies board distortion via multiple “rings.”
- FUN_00401f80: Displays the scrambled chessboard.

The XOR key computation and basic decoding are only setup phases for the central permutation challenge.

Step 2: Recognizing Multi-Ring Obfuscation

By reviewing FUN_00401d40 and its referenced arrays, the solver traces four separate index arrays in .rodata, each representing a set of squares forming concentric "rings" on the chessboard

The function references four distinct data arrays in memory:

- DAT_004a4130 – Ring 1 indices, starting at offset 0x004a4130
- DAT_004a4140 – Ring 2 indices, starting at offset 0x004a4140
- DAT_004a4180 – Ring 3 indices, starting at offset 0x004a4180
- DAT_004a41e0 – Ring 4 indices, starting at offset 0x004a41e0

Step 3: Extracting Ring Index Arrays

Ring Array Extraction from .rodata

Using Ghidra's memory viewer and cross-references, we extract the arrays referenced by the distortion function. Each array contains 4-byte little-endian integers representing board indices (0-63).

Ring 1 (at 0x004a4130, length 4):

Hex: 23 00 00 00 1b 00 00 00 1c 00 00 00 24 00 00 00

Decoded: [35, 27, 28, 36]

Ring 2 (at 0x004a4140, length 12):

Hex: 12 00 00 00 13 00 00 00 14 00 00 00 15 00 00 00

1d 00 00 00 25 00 00 00 2d 00 00 00 2c 00 00 00

2b 00 00 00 2a 00 00 00 22 00 00 00 1a 00 00 00

Decoded: [18, 19, 20, 21, 29, 37, 45, 44, 43, 42, 34, 26]

Ring 3 (at 0x004a4180, length 20)

Hex: 09 00 00 00 0a 00 00 00 0b 00 00 00 0c 00 00 00

0d 00 00 00 0e 00 00 00 16 00 00 00 1e 00 00 00

26 00 00 00 2e 00 00 00 36 00 00 00 35 00 00 00

34 00 00 00 33 00 00 00 32 00 00 00 31 00 00 00

29 00 00 00 21 00 00 00 19 00 00 00 11 00 00 00

Decoded: [9, 10, 11, 12, 13, 14, 22, 30, 38, 46, 54, 53, 52, 51, 50, 49, 41, 33, 25, 17]

Ring 4 (at 0x004a41e0, length 28):

Hex: 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00

04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00

0f 00 00 00 17 00 00 00 1f 00 00 00 27 00 00 00

2f 00 00 00 37 00 00 00 3f 00 00 00 3e 00 00 00

3d 00 00 00 3c 00 00 00 3b 00 00 00 3a 00 00 00

39 00 00 00 38 00 00 00 30 00 00 00 28 00 00 00

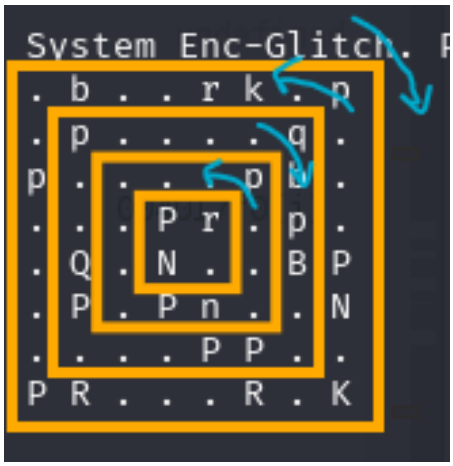
20 00 00 00 18 00 00 00 10 00 00 00 08 00 00 00

Decoded: [0, 1, 2, 3, 4, 5, 6, 7, 15, 23, 31, 39, 47, 55, 63, 62, 61, 60, 59, 58, 57, 56, 48, 40, 32, 24, 16, 8]

Step 4: Understanding Ring Rotation Mechanics

Visualizing Ring Structure

Mapping these indices onto an 8×8 chessboard (indices 0-63, row-major order):



These represent concentric rectangular rings spiraling from the outer edge toward the center, a classic permutation pattern.

Analyzing Rotation Directions

By tracing the loop structures in FUN_00401d40, we observe distinct patterns:

For each ring, the decompiler shows operations like:

- board[ring[(i + 1) % len(ring)]] = temp[i] → Clockwise rotation (shift right)
- board[ring[(i - 1 + len) % len(ring)]] = temp[i] → Counterclockwise rotation (shift left)

Through careful analysis of the actual loop implementations and their patterns, we determine:

- Ring 1: Rotated counterclockwise (-1)
- Ring 2: Rotated clockwise (+1)
- Ring 3: Rotated counterclockwise (-1)
- Ring 4: Rotated clockwise (+1)

Step 5: Building the Recovery Script

Reversal Strategy

To undo the distortion and recover the original board:

1. Apply rotations in reverse order (ring4 → ring3 → ring2 → ring1)
2. For each ring, apply the opposite direction:
    - If ring was rotated clockwise, reverse with counterclockwise
    - If ring was rotated counterclockwise, reverse with clockwise

Reversal mapping:

- Ring 4 was rotated clockwise → reverse with counterclockwise
- Ring 3 was rotated counterclockwise → reverse with clockwise
- Ring 2 was rotated clockwise → reverse with counterclockwise
- Ring 1 was rotated counterclockwise → reverse with clockwise

Python Recovery Script

```python
# === Paste your extracted ring indices ===
ring1 = [35, 27, 28, 36]
ring2 = [18, 19, 20, 21, 29, 37, 45, 44, 43, 42, 34, 26]
ring3 = [9, 10, 11, 12, 13, 14, 22, 30, 38, 46, 54, 53, 52, 51, 50, 49, 41, 33, 25, 17]
ring4 = [0, 1, 2, 3, 4, 5, 6, 7, 15, 23, 31, 39, 47, 55, 63, 62, 61, 60, 59, 58, 57, 56, 48, 40, 32, 24, 16, 8]

distorted_board = """
. b . . r k . p
. p . . . . q .
p . . . . p b .
. . . P r . p .
. Q . N . . B P
. P . P n . . N
. . . . P P . .
P R . . . R . K
"""

def parse_board(board_str):
    lines = [line for line in board_str.strip().split('\n') if line.strip()]
    board = []
    for line in lines:
        board.extend(line.split())
    return board

def rotate_ring_forward(board, ring):
    temp = [board[i] for i in ring]
    for i in range(len(ring)):
        board[ring[(i + 1) % len(ring)]] = temp[i]

def rotate_ring_backward(board, ring):
    temp = [board[i] for i in ring]
    for i in range(len(ring)):
        board[ring[(i - 1) % len(ring)]] = temp[i]

def undistort_board(board):
    # Reverse all ring rotations: reverse order and reverse direction
    rotate_ring_forward(board, ring4)      # Reverse -1 with +1
    rotate_ring_backward(board, ring3)     # Reverse +1 with -1
    rotate_ring_forward(board, ring2)      # Reverse -1 with +1
    rotate_ring_backward(board, ring1)     # Reverse +1 with -1
    return board

def print_board(board):
    for i in range(8):
        print(' '.join(board[i*8:(i+1)*8]))

if __name__ == "__main__":
    print("Distorted board:")
    original = parse_board(distorted_board)
    print_board(original)

    print("\nRecovered board:")
    recovered = undistort_board(original[:])
    print_board(recovered)
```

Recovered Board:

```
┌──(rigalis㉿Kali)-[~/Downloads]
└─$ python3 recover_board.py
Distorted board:
. b . . r k . p
. p . . . . q .
p . . . . p b .
. . . P r . p .
. Q . N . . B P
. P . P n . . N
. . . . P P . .
P R . . . R . K

Recovered board:
. . b . . r k .
p . . . . q b p
. p . . . . p .
. . . r . p B .
. . . P N . . .
. Q P n . . . P
P P . . . P P N
R . . . R . K .
```

Translating this position online on any Chess Analysis Board gives you this:

This position translates to the
FEN:2b2rk1/p4qbp/1p4p1/3r1pB1/3PN3/1QPn3P/PP3PPN/R3R1K1 w - - 0 1
White to move :)

As the engine suggests, the best move is Knight to f6, playing which

Brings us to this independent position:

2b2rk1/p4qbp/1p3Np1/3r1pB1/3P4/1QPn3P/PP3PPN/R3R1K1 b - - 1 1

Input this FEN in the binary:

Flag: DJSISACA{CH3SS_R3V_M4STER!!!}