FLAREON12(2025)

Drill Baby Drill!

Challenge Author: Nick Harbour

Writeup by scaredandalone

I ended up solving this problem without even opening the executable, since we have the python source code on hand.

If we look into the source, we can find the GenerateFlagText() function, which takes in a sum in order to generate a key and uses that key to decode some raw byte value.

Figure 1: GenerateFlagText() Function

```
def GenerateFlagText(sum):
    key = sum >> 8
    encoded = "\xd0\xc7\xdf\xdb\xd4\xd0\xd4\xdc\xe3\xdb\xd1\xcd\x9f\xb5\xa7\xa7\xa0\xac\xa3\xb4\x88\xaf\xa6\xaa\xbe\xa8\xe3\xa0\xbe\xff\xb1\xbc\xb9"
    plaintext = []
    for i in range(0, len(encoded)):
        plaintext.append(chr(ord(encoded[i]) ^ (key+i)))
    return ''.join(plaintext)
```

Since the key is simply just the value of the sum, shifted right by 8 bits we can generate the key if we figure out how the sum is calculated.

Figure 2: Reference to the flag gen function

```
# draw the baby
player.draw(screen)

if player.hitBoulder():
    boulder_mode = True

if player.hitBear():
    player.drill.retract()
    bear_sum *= player.x
    bear_mode = True

if bear_mode:
    screen.blit(bearimage, (player.rect.x, screen_height - tile_size))
    if current_level == len(LevelNames) - 1 and not victory_mode:
        victory_mode = True
        flag_text = GenerateFlagText(bear_sum)
        print("Your Flag: " + flag_text)
```

Now it becomes obvious what is going on.

When the player hits a bear with their drill, the “bear_sum” value is multiplied by the player.x (x coordinate) of the player and stored back into the bear_sum.

We can see that the bear_sum is being used to generate the flag text, so we need to just figure out the x coordinate of each bear and use the product of all positions in order to get a valid sum for the flag decryption.

To do this we need to figure out how the bears are being placed in each level (and where).

Figure 3: Boulder generation code

```
boulder_layout = []
for i in range(0, tiles_width):
    if (i != len(LevelNames[current_level])):
        boulder_layout.append(random.randint(2, max_drill_level))
    else:
        boulder_layout.append(-1)
```

In this game, a boulder signifies a column which is “incorrect” (meaning there's no bear in it). Looking carefully, we can see that we can easily discern which column is “safe” or “correct” just by getting the length of the current level's name, and digging down that column.

Figure 4: LevelNames

```
LevelNames = [
    'California',
    'Ohio',
    'Death Valley',
    'Mexico',
    'The Grand Canyon'
]
```

For example, in the first level “California” the safe column is column 10, “Ohio” is column 4, etc. This means that the bear is in the column which equals the length of the level name.
Now since the bear_sum is just the product of all the x positions that the teddies are in, we can just multiply the lengths of all levels together to get a valid sum.

bear_sum = 10 * 4 * 12 * 6 * 16 = 46080
key = bear_sum >> 8

Decoding the flag, we get:

drilling_for_teddies@flare-on.com