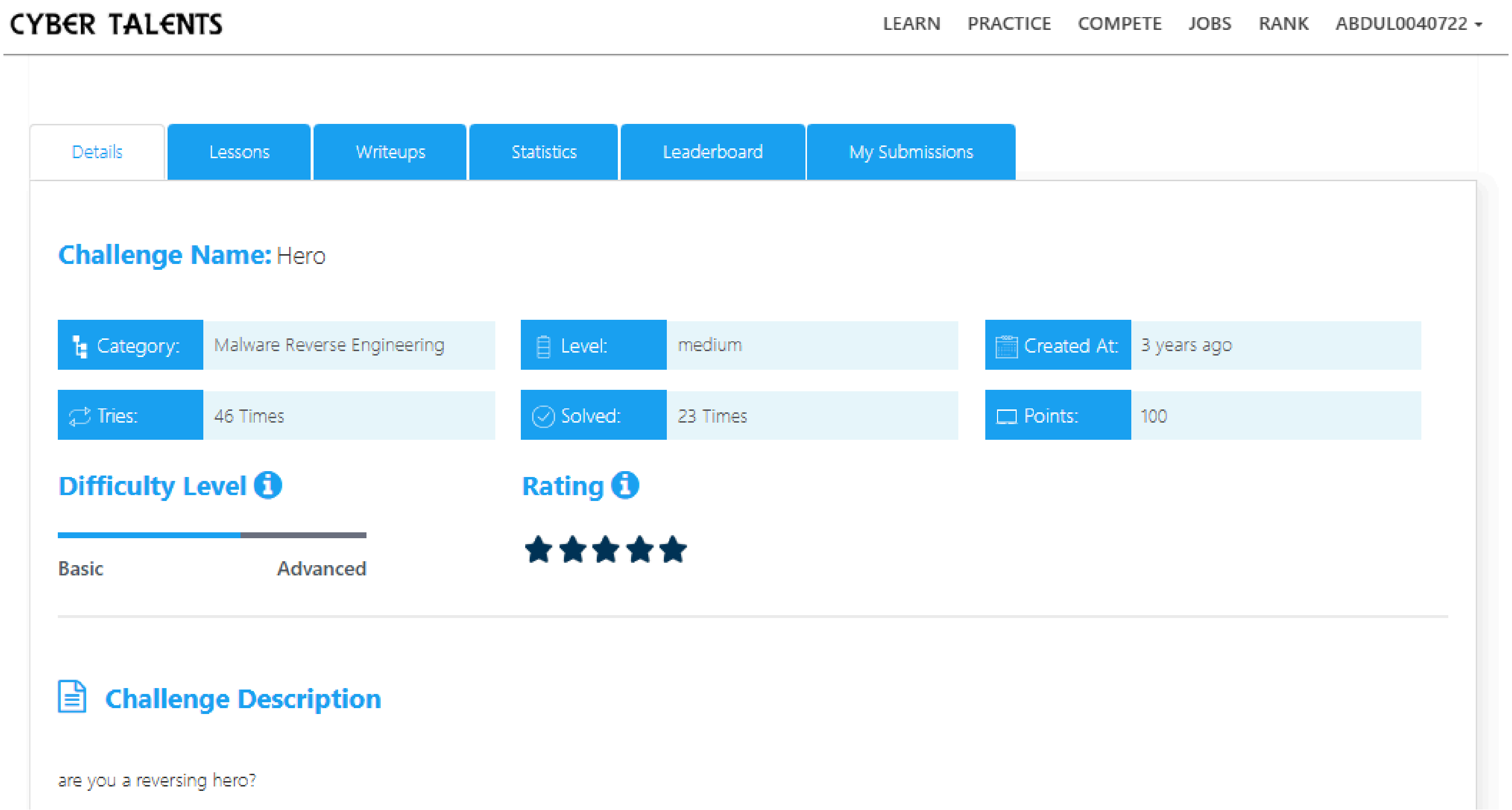
Hero



CYBER TALENTS

LEARN PRACTICE COMPETE JOBS RANK ABDUL0040722

Details | Lessons | Writeups | Statistics | Leaderboard | My Submissions

Challenge Name: Hero

Category:	Malware Reverse Engineering	Level:	medium	Created At:	3 years ago
Tries:	46 Times	Solved:	23 Times	Points:	100

Difficulty Level

Basic — Advanced

Rating

★★★★★

Challenge Description

are you a reversing hero?

Attached files:

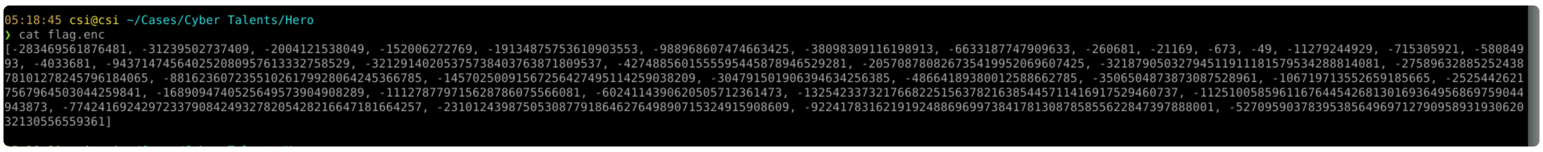
`Hero.unknown` — some form of unknown ASCII text (implied to be disassembled code)

`flag.enc` — a JSON file with a long list of large negative integers.

Initial Recon

```
$ file Hero.unknown
Hero.unknown: ASCII text

$ file flag.enc
flag.enc: JSON data
```

Peeking inside:



```
05:18:45 csi@csi ~/Cases/Cyber Talents/Hero
❯ cat flag.enc
[-283469561876481, -31239502737409, -2004121538049, -152006272769, -19134875753610903553, -988968607474663425, -38098309116198913, -6633187747909633, -260681, -21169, -673, -49, -11279244929, -715305921, -580849
93, -4033681, -943714745640252080957613332758529, -32129140205375738403763871809537, -4274885601555595445878946529281, -205708780826735419952069607425, -32187905032794511911181579534288814081, -27589632885252438
78101278245796184065, -88162360723551026179928064245366785, -14570250091567256427495114259038209, -304791501906394634256385, -48664189380012588662785, -3506504873873087528961, -106719713552659185665, -2525442621
7567964503044259841, -1689094740525649573904908289, -111278779715628786075566081, -6024114390620505712361473, -1325423373217668225156378216385445711416917529460737, -112510058596116764454268130169364956869759044
943873, -7742416924297233790842493278205428216647181664257, -231012439875053087791864627649890715324915908609, -92241783162191924886969973841781308785855622847397888001, -5270959037839538564969712790958931930620
32130556559361]
```

Then

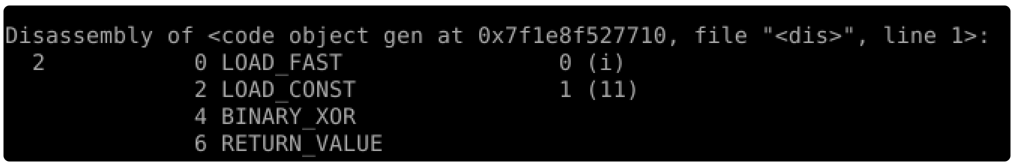
```
cat Hero.unknown
```

That last part was the clue that gave it away. I recognized the structure: `LOAD_CONST`, `MAKE_FUNCTION`, etc… It was disassembled Python bytecode, likely produced with the `dis` module. So the actual obfuscation logic was right there. All we needed to do was understand what it was doing, then invert it.

Understanding `Hero.unknown`

1. Function Definitions

`gen(i): return i ^ 11`

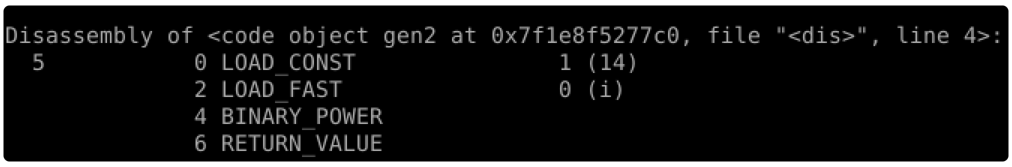


```
Disassembly of <code object gen at 0x7f1e8f527710, file "<dis>", line 1>:
  2           0 LOAD_FAST                0 (i)
              2 LOAD_CONST               1 (11)
              4 BINARY_XOR
              6 RETURN_VALUE
```

This means:

- Load argument `i`
- Load constant `11`
- XOR them → `i ^ 11`
- Return result

`gen2(i): return 14 ** i`



```
Disassembly of <code object gen2 at 0x7f1e8f5277c0, file "<dis>", line 4>:
  5           0 LOAD_CONST               1 (14)
              2 LOAD_FAST                0 (i)
              4 BINARY_POWER
              6 RETURN_VALUE
```

This means:

- Load constant `14`
- Load argument `i`
- Compute `14 ** i`
- Return result

2. Main Routine

Now parse the bytecode of the main execution step by step.

Lines 1–6: Function Definitions

```
1           0 LOAD_CONST               0 (<code object gen at 0x7f1e8f527710, file "<dis>", line 1>)
            2 LOAD_CONST               1 ('gen')
            4 MAKE_FUNCTION            0
            6 STORE_NAME               0 (gen)
```

Creates and stores function `gen`.

```
 4           8 LOAD_CONST               2 (<code object gen2 at 0x7f1e8f5277c0, file "<dis>", line 4>)
            10 LOAD_CONST               3 ('gen2')
            12 MAKE_FUNCTION            0
            14 STORE_NAME               1 (gen2)
```

Creates and stores function `gen2`.

Lines 7–9: Read flag.txt and convert to list of ASCII values

```
 7          16 LOAD_NAME                2 (open)
            18 LOAD_CONST               4 ('flag.txt')
            20 LOAD_CONST               5 ('r')
            22 CALL_FUNCTION            2
            24 STORE_NAME               3 (f)
```

Opens `flag.txt` for reading, stores file object as `f`.

```
8          26 BUILD_LIST               0
           28 STORE_NAME               4 (o)
```

Initializes empty list `o`.

```
30 LOAD_NAME                3 (f)
32 LOAD_METHOD              5 (readlines)
34 CALL_METHOD              0
36 LOAD_CONST               6 (0)
38 BINARY_SUBSCR
40 STORE_NAME               6 (r)
```

Reads all lines from the file → `f.readlines()`, then grabs the first line with `[0]` → stored as `r`.

```
10          42 LOAD_NAME                7 (range)
            44 LOAD_NAME                8 (len)
            46 LOAD_NAME                6 (r)
            48 CALL_FUNCTION            1
            50 CALL_FUNCTION            1
            52 GET_ITER
       >>   54 FOR_ITER                22 (to 78)
            56 STORE_NAME               9 (i)
```

For loop over each index `i` in `range(len(r))`.

```
12          58 LOAD_NAME                4 (o)
            60 LOAD_METHOD             10 (append)
            62 LOAD_NAME               11 (ord)
            64 LOAD_NAME                6 (r)
            66 LOAD_NAME                9 (i)
            68 BINARY_SUBSCR
            70 CALL_FUNCTION            1
            72 CALL_METHOD              1
            74 POP_TOP
            76 JUMP_ABSOLUTE           54
```

For each character at `r[i]`:

- Get its ASCII value with `ord(r[i])`
- Append to list `o`

➡ So now `o = [ord(c) for c in r]`

What's Happening So Far:

- Read first line from `flag.txt`
- Convert each character into its ASCII value
- Store in list `o`

Lines 14–20: Obfuscation

```
14     >>   78 BUILD_LIST                0
            80 STORE_NAME               12 (s)
```

Creates a new empty list `s` (output list of encrypted values).

```
15          82 LOAD_NAME                 7 (range)
            84 LOAD_NAME                 8 (len)
            86 LOAD_NAME                 4 (o)
            88 CALL_FUNCTION             1
            90 CALL_FUNCTION             1
            92 GET_ITER
       >>   94 FOR_ITER                 40 (to 136)
            96 STORE_NAME                9 (i)
```

Loop over each index `i` in `range(len(o))`

Inside the loop:

```
16          98 LOAD_NAME                 0 (gen)
           100 LOAD_NAME                 9 (i)
           102 CALL_FUNCTION             1
           104 STORE_NAME               13 (t)
```

`t = gen(i)` → XOR `i ^ 11`

```
17         106 LOAD_NAME                 1 (gen2)
           108 LOAD_NAME                13 (t)
           110 CALL_FUNCTION             1
           112 STORE_NAME                3 (f)
```

`f = gen2(t)` → `14 ** (i ^ 11)`

```
18         114 LOAD_NAME                12 (s)
           116 LOAD_METHOD              10 (append)
           118 LOAD_NAME                 3 (f)
           120 LOAD_NAME                 4 (o)
           122 LOAD_NAME                 9 (i)
           124 BINARY_SUBSCR
           126 BINARY_MULTIPLY
           128 UNARY_INVERT
           130 CALL_METHOD               1
           132 POP_TOP
           134 JUMP_ABSOLUTE            94
```

Now comes the obfuscation:

```
val = f * o[i]         # Multiply 14^(i ^ 11) * ord(flag[i])
val = ~val             # Bitwise NOT
s.append(val)
```

This is how the `flag.enc` values were created.

Line 20–21: Output

```
20     >>  136 LOAD_NAME                14 (print)
           138 LOAD_NAME                12 (s)
           140 CALL_FUNCTION             1
           142 POP_TOP

21         144 LOAD_NAME                14 (print)
           146 LOAD_NAME                 8 (len)
           148 LOAD_NAME                12 (s)
           150 CALL_FUNCTION             1
           152 CALL_FUNCTION             1
           154 POP_TOP
```

Just prints `s` and its length. Not important to us.

Final Formula

With this understanding, the logic of the encryption is:

```
for i, c in enumerate(flag_line):
    t = i ^ 11
    f = 14 ** t
    encrypted = ~(f * ord(c))
    s.append(encrypted)
```

So to reverse it:

```
original_char = chr((~encrypted_value) // (14 ** (i ^ 11)))
```

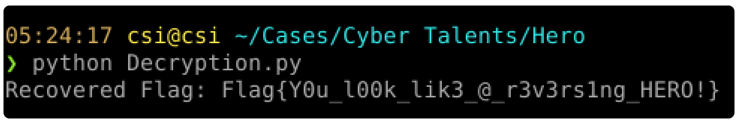
And that's how I got the flag back.

🛠 Reversing the Obfuscation

Here's the final script I wrote to undo it:

```python
import json

with open('flag.enc', 'r') as f:
    encrypted = json.load(f)

def gen(i):
    return i ^ 11

def gen2(i):
    return 14 ** i

flag_chars = []

for i, val in enumerate(encrypted):
    power = gen2(gen(i))
    ch = (~val) // power
    flag_chars.append(chr(ch))

flag = ''.join(flag_chars)
print("Recovered Flag:", flag)
```

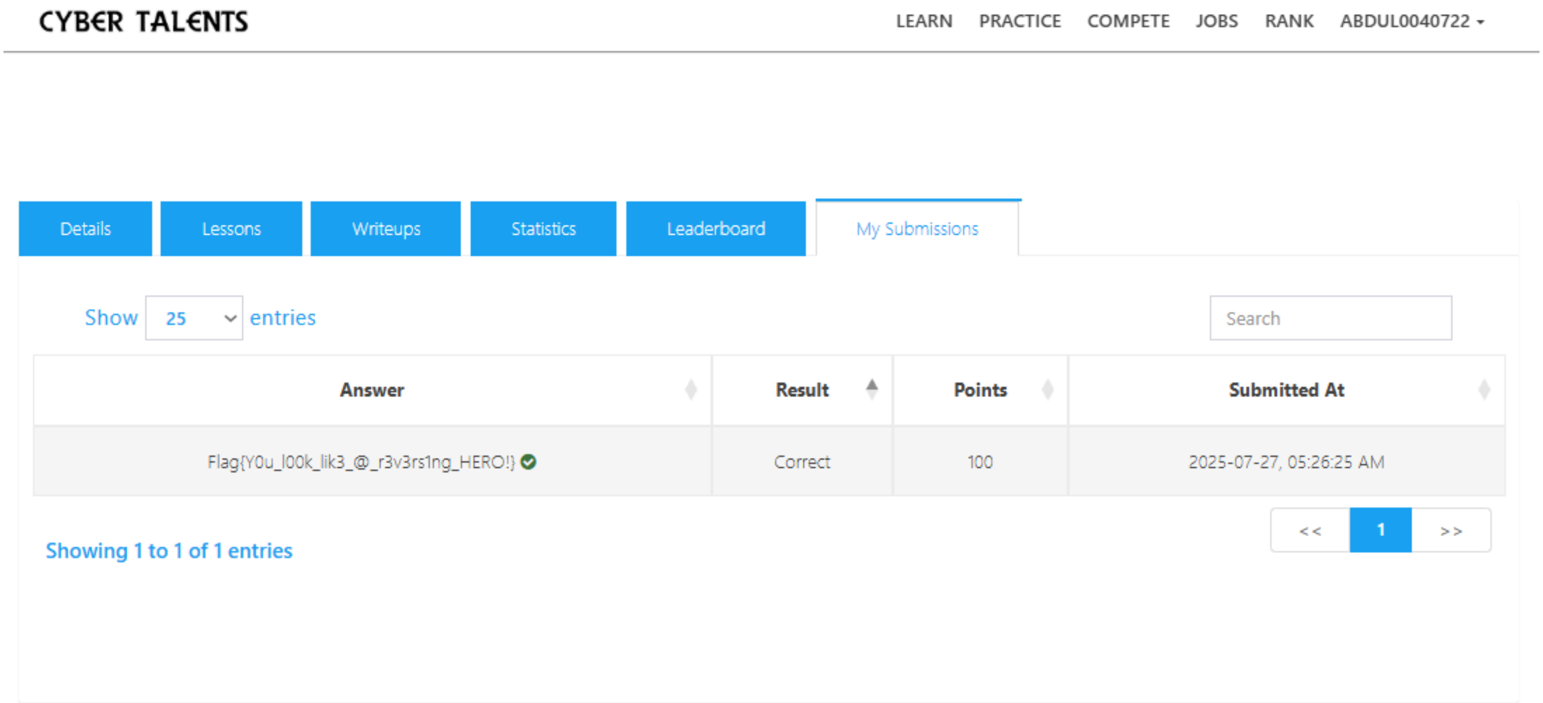
Output



```
05:24:17 csi@csi ~/Cases/Cyber Talents/Hero
❯ python Decryption.py
Recovered Flag: Flag{Y0u_l00k_lik3_@_r3v3rs1ng_HERO!}
```

```
Recovered Flag: Flag{Y0u_l00k_lik3_@_r3v3rs1ng_HERO!}
```

And there it was — clear as day.



Answer	Result	Points	Submitted At
Flag{Y0u_l00k_lik3_@_r3v3rs1ng_HERO!} ✓	Correct	100	2025-07-27, 05:26:25 AM

Showing 1 to 1 of 1 entries